Journal of Industrial Technology 5 (1), 1995, 65-74

SOFTWARE REUSABILITY: EXAMPLES, PROBLEMS AND STRATEGIES

Elizabeth Key and Khairuddin Hashim

Faculty of Computer Science and Information Technology
University of Malaya,
59100 Kuala Lumpur, Malaysia

RINGKASAN: *Penggunaan semula perisian telah di lihat sebagai suatu penyelesaian yang mempunyai harapan untuk menyelesaikan masalah-masalah semasa didalam pembangunan perisian seperti sistem-sistem perisian yang tidak lengkap dan lewat disiapkan. Guna semula perisian meningkatkan produktiviti dan kualiti disamping menyingkatkan masa pembangunan. Kertaskerja ini membincangkan isu-isu berkaitan penggunaan semula perisian dengan contoh-contoh yang telah berjaya, masalah-masalah dan strategi-strategi semasa pelaksanaannya.*

ABSTRACT: Software reusability has been seen as a very promising solution to solving current software development problems such as, incomplete and late delivery of a given job. It increases productivity and quality while shortening development time. This paper discusses issues related to adopting the reuse of software with some successful examples, problems and strategies involved in its implementation.

KEYWORDS: Software, software reusability, software development.

INTRODUCTION

During the past 40 years, computer hardware has taken a gigantic step, a step which was believed impossible. The advances in microelectronics throughout the 80's has brought about a tremendous increase in computing power at a much lower cost. However, all these fast and powerful hardware chips are unable to satisfy the rising demand for new software. The 1990's aims to harness and tap the potential of these modern hardware facilities by improving software quality and its productivity. Musa (1985) estimated that the demand for software has been increasing by 900% for each decade. However, the productivity of individual software engineers increases by only 35% which accounts for the high cost in software production.

To meet this high continuous demand for new software applications, a factor contributing to the hardware progress: the reuse of previously developed components in the new hardware systems concept is imported to software production. Software components viewed upon as software "chips" are depicted in Figure 1. These can be reused in new systems and be easily plugged into a new software system's socket thus reducing software development time and effort.

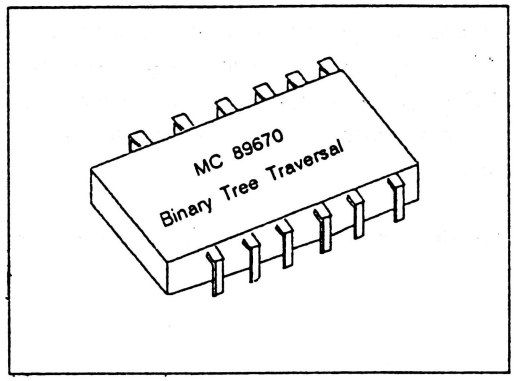

Figure 1. *A software component*

SOFTWARE DEVELOPMENT PROBLEMS

Software is now becoming the infrastructure of companies, giving rise to its continuous demand. Meeting such demands is not just by simply churning out software. Some software development problems (Pressman, 1992) are given below:

a) Schedule and cost estimates are often grossly inaccurate
b) Productivity of the software people is not in pace with the demand for their services, and
c) Quality of software is often less than adequate.

These software development problems usher in the "software quality and productivity era". The 1990's hope to curtail this "software crisis" via some breakthroughs in software development approaches and technologies and at the same time cut down the cost of software development.

AN APPROACH TO OVERCOME SOFTWARE DEVELOPMENT PROBLEMS

The existing gap between the demand and the productivity ability calls for improved software development technologies. One approach that could significantly contribute to high quality and productivity is the reuse oriented development technology. Some reuse strategies would be: automating software production, delegating software production to an expert system and reusing existing software (Ng, 1990), each strategy being reuse-centred.

Figure 2 indicates that the problem of reducing software cost can be approached through components reuse and software design reuse.

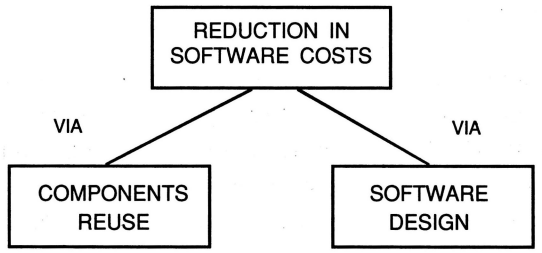

Figure 2. *Reducing software development costs*

THE REUSE NOTION

The notion of reuse and reworking software in software development has its beginnings in the 1950s. Reuse can be differentiated into the 'black-box' reuse and the 'white-box' reuse (Shriver, 1987). 'Black-box' reuse refers to using an entity in a different context from that in which it was originally used whereas 'white-box' reuse or rework would mean that the entity is modified before it is being used.

Reuse stops the continual reinvention of code for common functions and improves on the code quality through the use of debugged modules (Gruman, 1988). The incentives are largely economical, centred on the development and maintenance sector; especially for large and complex projects.

Reusability saves a lot of coding, such as in complicated function libraries. Quality is assured at minimal cost because the components have been debugged extensively during development.

Merits and Promises of Reuse

In spite of the fears and risks involved, systematic software reuse do offer a number of advantages:

1. System reliability is increased as reused components are adequately tested component and should be more reliable than components developed anew.
2. The overall risk is reduced. The existing component indicates its reliability thus reducing the uncertainty in the costs of using it than the costs of developing it.
3. Overcomes the fears that there will be too few programmers to handle the workload due to the role of software in the infrastructure of a business. Moreover, character to character 'replay' by specialists with different project groups cannot meet software demands. Such manpower can be harnessed more effectively through the encapsulation of their knowledge in reusable components development.
4. Organization standards can be embodied in reusable components.
5. Software development time can be reduced.
6. Software development cost is reduced.
7. Productivity is increased.

Raytheon's Missile System Division of Information Processing Systems Organization has developed a breakthrough approach in developing and maintaining their software (Lanergan and Grasso, 1984). This reusability approach was adopted realising the fact that 60 percent of all business application designs and codes are redundant and could be standardized and reused hence resulting in significant gains in productivity and reliability.

By standardizing functions in the form of reusable functional modules and logic structures, a 50 percent gain in productivity can be attained. Data processing organization can redeploy 60-80 percent of their resources to work on new systems development applications.

Japanese software factories have reported great improvement in programmers' productivity through reusability by integrating known techniques from different disciplines like resource management, production engineering, quality control, software engineering and industrial psychology (Tajima and Matsubara, 1984).

Moreover, AT&T Bell Laboratories in New Jersey reported success in a three-year reuse approach program (Gruman, 1988). Reusing components tripled its productivity. In two transaction-processing projects, one with 120,000 and the other with 210,000 lines of codes, reuse rates were 65 percent. In a 1.2 million-line project, 86 percent were reused.

In another example, metrics collected in two reuse programs at Hewlett-Packard (Lim, 1994) demontrated improved quality, increased productivity, and reduced marketing time. Chen and Lee (1993) described an experiment that substantiated a proposed software reuse approach using reusable C++ components in terms of software productivity and quality.

PROBLEMS IN SOFTWARE REUSE

Despite its promises, software reuse is not widely practised. Software reuse at Motorola (Joss, 1994) caused high initial cost and slow return on investment. Top management intervention was necessary to proceed and reap the benefits of reuse. In reuse, one hopes to build on someone else's successes, however one could also inherit their mistakes or find out that the reuse of "as-is" is not that simple. This calls for the measuring of software quality in reuse software to avoid the churning out of reuseless software. Codes must be designed for reuse in the first place. However, such codes are difficult and costly to reuse. Reusable codes might cost 30 to 200 percent more to develop, document, and test, but subsequent reuse costs 20 to 40 percent less than that from scratch writing (Traz, 1988).

Some reasons for the lack of enthusiasm towards software reuse are given below (Biggerstaff and Richer, 1987):

1. The critical component attributes for reusable components has yet to be identified. Factors contributing to the reusability of a component must be determined.
2. To meet the criteria of reusability, generalized components which are expensive must be developed. The increase in project cost is not favourable for managers who try to bring down project costs.
3. Many developers are reluctant to accept "canned" components, probably due to the lack of confidence in other developers and prefer to write components afresh.
4. Software reuse would mean a good classifying scheme for components to be stored in a repository. Cataloguing and retrieving software components from the repository would again, require initial investment, and many developers are reluctant to invest fearing the risk involved.

DILEMMA IN REUSE

The main dilemma is the antagonistic nature of the factors governing reusability.

Generality of Applicability Versus Pay-off

Comparing the relative characteristic of various technologies, pay-off from general technologies is lower as compared to more specific technologies (Joos, 1994).

Component Size Versus Reuse Potential

New parameters and options added to a component so as to increase its domain of applicability ends up increasing its size making it more complex. Components with a higher degree of complexity results in lower comprehensability and incurs higher maintenance

overheads. Moreover, creating such robust interfaces and maintaining those general and generic components requires the insight from seeing how software has been used in the past and envisioning how it might be used in the future.

Another aspect of component growth can be seen when components becomes more and more specific narrowing its applications. Such components may perform excellently in its own context in which it was designed for, but taken out from the specific domain it would increase reusing cost due to modification or rather becomes reuseless.

The Cost of Library Population

To reap the fruits of reusability, a great deal of intellectual capital, real capital and time must be invested.

BARRIERS TO SOFTWARE REUSE

Barriers are not insurmountable. However to surmount them (Traz, 1988):

1. Useful and certified components must be made available.
2. Tools and methods to support both creation and use of component libraries must be developed.
3. Money, time and personnel must be expended to develop software reuse and train users to become proficient in them.

From the software productivity perspective profitability depends largely on the availability of certified component and on easily locating components, assessing their applicability and incorporating them into the software system being developed. Certified components refer to components of a high degree of reusability.

The golden rules of reusability are:

- "Before you can reuse something, you have to find it"
- "Before you can reuse it, you have to know what it does"
- "Before you can reuse something, you have to know how to reuse it"

Abiding by the golden rules of reusability, the focus is on some crucial issues of reusability as listed below:

(i) representation of components
(ii) qualifying components
(iii) building a library scheme
(iv) tools aiding modifying and incorporating components into the new software system

APPROACHES TO COST-EFFECTIVE SOFTWARE REUSE

An expression for the relationship between reuse-investment is through the definition of the quality-of-investment measure (Barnes and Bollinger, 1991), Q; a ratio of reuse benefits to reuse investments, is defined as:

$$Q = B/R$$

where R is the total reuse investment and B is the total cost benefits resulting from the reuse investment.

Details on cost-effective software reuse has been shown by Biggerstaff and Richer, (1987). The key to increase reuse cost-effectiveness is to increase the Q measure. This can be accomplished by three strategies as listed below:

(i) Increase the level of consumer reuse
(ii) Reduce the average cost of reuse
(iii) Reduce the investment needed to achieve a given reuse benefit

Increasing the Level of Consumer Reuse

The demand for reusable components is highly dependable on the intrinsic quality of the components. Components of high degree of reusability, created and designed with reuse in mind are likely to have a larger reuse market than components lacking such features. A reuse instance occurs when a component is actually being reused in the development activity of the new system. The strategy is to keep the average level of reuse investment per unit of code low if the expected number of reuse instance is low and vice versa. In other words, if B is small, R should be as low as possible as Q increases linearly to B.

Reducing Cost of Reuse

From the investment-quality measure, Q = B/R, the next technique to increase reuse-cost effectiveness would be to reduce R. Reuse producers need to promote reuse products by making them easily available, adaptable and able to integrate into the new system. Investments in such efforts is dependent on the expected levels of reuse and the availability of appropriate reuse technologies. Reducing cost of reuse depends on a large degree on selecting technologies that adequately support the needs of reuse customers.

Reusable components can be put in their perspective in a three dimensional space with the following axes:

* Reuse investment costs
* Component generality
* Cost to reuse

Reuse investment costs refers to the total cost incurred by the producer to make parts readily available for reuse. A component's generality gives a good indication of its reusable power in terms of its available variations. Cost to reuse is the total cost to the reuser for finding, adapting, integrating and testing a reusable component. By striking a balance between the three aspects: to identify a plausible level of reuse investment with a corresponding reuse technology, the cost to reuse can be minimized.

Reducing Investment Costs

To reduce investment costs, the level of generality provided by the reuse technologies matching future expected needs must be identified. Components must be designed with foresight to meet the future needs. Another approach to reduce investment cost is not to adhere to one reuse technology for a given set of components. A set of technologies must be chosen for the best and economical software construction.

REUSE STRATEGIES

One golden rule to follow when designing components is to ensure its modularity. Modularization controls the size and complexity of components making it highly reusable. With the ideal granularity, components are now available for reuse. However, before reuse, a developer would consider its nature or function. There are two fundamental forms of functionality (Biggerstaff and Richerm, 1987).

(i) Invariant functionality
(ii) Variant functionality

Invariant functionality forms the core of software construction and usually includes mathematical functions. Variant components are those that evolved with software requirements meeting the current needs of reusers. Mixing both the variant and invariant functionality types makes it a good strategy for reuse. Two approaches in reusing variant and invariant functionality types as suggested by Biggerstaff and Richer (1987) are adaptive reuse and compositional reuse.

Adaptive reuse uses large structures as invariants and restricts variability to low levels while compositional reuse uses invariants and variant functionality to link modules together. A hybrid approach would be an ideal strategy where parts of systems can be retrieved from a reuse library to be used 'as is' or components retrieved can be modified. This strategy is implemented to overcome some of the problems in software reuse. This approach hopes to give some insight into software reuse: reuse as a mechanism for preserving and "canning" human creativity and ingenuity.

Components suitable for reuse must be self-contained having features contributing to its reusability. Basically, components can be reused either in a toolkit or by applying the general

approach, where existing components are put together by means of a command language or by inserting a component with satisfactory characteristics into a new application.

Mechanisms in Software Component Reuse

Before the fruits of software reuse can be reaped, much groundwork has to be done. The software reuse steps (Jones, 1990) are outlined as in Figure 3.

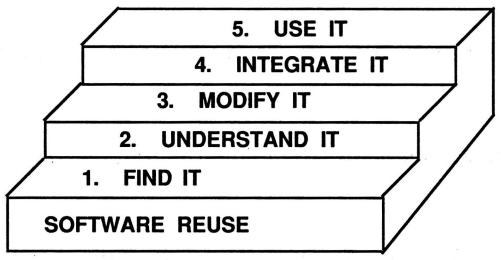

Figure 3. *Steps in software reuse*

A more detailed activity of software component reuse are as follows:

a) Defining attributes contributing to the reusability of software components.
b) Profiling software components.
c) Qualifying software components for reuse.
d) Classifying reusable software components.
e) Establishing a browse structure.
f) Designing an effective retrieval system.

CONCLUSIONS

Software reuse shows promises not only in software development costs reduction but also in upgrading software quality and increasing software productivity. Investments in software reuse would allow reusers to reap the fruits of reuse.

REFERENCES

Barnes, B.H. and Bollinger, T.B. (1991). Making reuse cost-effective, *IEEE Software,* **January,** 13-24.

Biggerstaff, T. and Richer, C. (1987). Reusability framework, assessment and directions, *IEEE Software*, **March**, 41-49.

Chen, D. and Lee, P.J. (1993). On the study of software reuse using reusable C++ Components, *J. Systems Sci.* **20**, 19-36.

Gruman, G. (1988). Soft News, *IEEE Software*, **November,** 87.

Jones, G.W. (1990). In: *Software Engineering,* Wiley & Sons, New York.

Joos, R. (1994). Software Reuse at Motorola, *IEEE Software*, **September**, 42-47.

Lanergan, R.G. and Grasso, C.A. (1984). Software engineering with reusable designs and code, *IEEE Trans. Software Eng.,* **SE-10 (5)**, 498-501.

Lim, W.C. (1994). Effects of reuse on quality, productivity, and economics, *IEEE Software*, **September,** 23-30.

Musa, J.D. (1985). Software Engineering: The future of a profession, *IEEE Software*, **22(1)**, 55-62.

Ng, Yeh (1990). In: *Modern Software Engineering - Foundation and Current Perspective*, Van Reinhold; New York.

Pressman, R.S. (1992). In: *Software Engineering: A practitioner's approach*, McGraw-Hill.

Shriver, B.D. (1987). Reuse revisited, *IEEE Software*, **July,** 5.

Tajima, D. and Matsubara, T. (1984). Inside the Japanese software industry, *Computer,* **17(3),** 34-43.

Will, T., Confessions of a used-program salesman - reuseless software, *Computer*, **December,** 75.

Woodfield, S.N., Embley, D.W. and Scott, D.T. Can programmers reuse software?, *IEEE Software*, **July,** 52-59.

ORDER FORM

JOURNAL OF INDUSTRIAL TECHNOLOGY
JURNAL TEKNOLOGI PERINDUSTRIAN

Please send to :

The Executive Editor
Journal of Industrial Technology
Block 15
SIRIM
P.O. Box 7035
40911 Shah Alam
Selangor.

SUBSCRIPTION PER ISSUE :
(inclusive of postage)
Malaysia RM 20
Overseas US$ 15.00
Delivery is by Second Class Airmail

PAYMENT :
To be sent with order.
Crossed Cheques/Bank Drafts/Money Orders/Postal Orders made payable to SIRIM.

Name : _____

Address: _____

_____

_____

Enclosed is Cheque/Bank Draft/Money Order/Postal Order No. _____ for RM/US$ _____ for ____ issue(s) beginning from _____

Date: _____ Signature : _____

JOURNAL OF INDUSTRIAL TECHNOLOGY

SCOPE

The Journal of Industrial Technology is published twice a year by the Standards and Industrial Research Institute of Malaysia (SIRIM). It reports on research undertaken in the following areas: Materials Science, Chemical Technology, Metal & Metalworking, Advanced Manufacturing Technology, Instrumentation & Measurement, Environmental Technology, Industrial Biotechnology, Industrial & Engineering Design and Product & Machine Development.

CONTRIBUTIONS

Original contributions related to the above specified technology areas are welcome on the strict understanding that the material has not been, nor is being considered for publication elsewhere. Research papers are to be submitted either in English or Bahasa Malaysia but abstracts in both languages should be given. Short communications are also welcome. Review articles may be published from time to time. While these are normally solicited by invitation, intending authors may also seek the agreement of the Chief Editor with respect to the proposed topic of discussion. Papers accepted become the copyright of the Journal and may not be printed or published in translation without the permission of the Chief Editor.

GENERAL INFORMATION

A diskette written in WordPerfect 5.0/5.1 (Windows)/WordStar 6.0/Aldus PageMaker 3.01 and three copies of the manuscript should be submitted (the original and two copies). Articles should be between 4,000 to 7,000 words (with the exception of review papers) and typewritten double-spaced on one side only of A4 paper and with a margin of about 40 mm all round. Words to be printed in italics should be written as such or underlined.

FORMAT AND STYLE

1. **Title page** (separate page). The title should be concise, descriptive and preferably not exceed 15 words. The name (s) of the author(s), affiliation (s) and full address (es) should be included.

2. **Abstract.** The abstract should precede the article and, in approximately 150 words, outline briefly the objectives and main conclusions of the paper. It should be followed by 10 keywords (separated by commas) identifying the matter for retrieval systems.

3. **Text.** The following style is intended essentially for scientific research papers. Engineering research papers may be written with appropriate headings and sub-headings.

   Introduction. The introduction should describe briefly the area of study and may give an outline of previous studies with supporting references. It should also indicate clearly the objectives of the paper.

   Materials and Methods. The materials used, the procedures followed with special reference to experimental design and analysis of data should be included.

   Results. Data of significant interest should be included. Tables and figures should be cited consecutively in the text with Arabic numerals. Do not intersperse tables and figures in text.

   Discussion. The contribution of the work to the overall knowledge of the subject could be shown. Further studies may also be projected.

   Conclusions. Conclusions derived from the study should normally be included.

   Acknowledgements. Brief thanks (no social or academic titles) or financial aid could be given in this section.

   References. References in the text should be denoted by giving the name(s) of the author(s) with year of publication in parentheses. Unpublished data or private communication should not appear in the reference list.

   In the list of references the general style is as follows:

   For journals.
   Lopez-Avila, V., Wesselman, R. and Edgell, K. (1990). Gas chromatographic - electron capture detection method for determination of 29 organochlorine pesticides in finished drinking water. Collaborative study. *J. Assoc. Off. Anal. Chem.,* **73(2)**, 276-289.

   For books.
   Thompson, D.P. and Korgul, P.(1983). In: *Progress in Nitrogen Ceramics,* ed. Riley, F.L., Martin Nijhoff, The Hague, The Netherlands, pp 375-380.

   For reports.
   Dubin, F.S., Mindell, H.L. and Bloome, S. (1976). *How to save energy and cut costs in existing industrial and commercial buildings.* An energy conservation report, May, Noyes Data Corporation, Park Ridge, USA.

   For proceedings.
   Siddiqi, S.A., Higgins, J. and Hendry, A. (1986). Production of sialon by carbothermal reaction of clay. In: *Proceedings of the International Conference on Non-oxide Technical and Engineering Ceramics,* ed. Hampshire, S., Elsevier Applied Science, London, pp 119-20.

4. **Illustrations.** All illustrations should be clearly drawn in Indian ink or should be photographed in sharp black and white, high-contrast, glossy prints. Illustrations should be planned to stand at least 50% reduction and be numbered consecutively in the same order as in the text, where they should be referred as "Figure" and not "Fig.". It could help the Editor if the authors could indicate as a marginal note where each illustration should be located in the text. Each illustration should also have a caption which not only explains the picture but also gives the title of the article which it illustrates. It is suggested that these captions be typewritten on a separate manuscript page. Each illustration must be clearly numbered and author's name be lightly written on the reverse side and assembled at the end of manuscript. Coloured photographs should only be included where they are essential in the paper.

5. **Tables** (one per page). They should be as concise as possible and not larger than a journal page. A descriptive title should be included so that the table does not need reference to the text. Abbreviate freely; if necessary explain in footnotes.

6. **Footnotes.** Footnoted information should only be used if it is absolutely necessary. The footnote should be typewritten on the bottom margin of the page where it should appear and an appropriate mark be designated on the text itself.

7. **Appendices.** Appendices should be typewritten on separate manuscript pages. Each should have a title and should be numbered consecutively.

8. **Units, symbols, abbreviations and conventions.** Units, symbols, abbreviations and conventions must follow the Systeme International d' Unites (S.I. units). Where non-standard abbreviations are used, the word(s) to be abbreviated should be written out in full on the first mention, followed by the abbreviation in parentheses.

   The Editor reserves the right to make literary corrections and to make suggestions to improve brevity.

REFEREES

All manuscripts will be refereed for relevance and quality. Authors may suggest in a covering letter the names of two qualified reviewers, i.e., individuals engaged in or versed in research of the type reported.

PROOFS

One set of proofs will be sent to the main author to be checked for printer's errors, and it is the responsibility of the author to submit corrections to the Editor.

REPRINTS

Authors will be provided with 20 free reprints of the article after publication.