PRODUCTIVE MAINTAINABILITY

Manuel J. Barranco García
Juan Carlos Granja Alvarez
Grupo de Lenguajes y Sistemas Informáticos e Ingeniería del Software
(P.A.I. 7060) Universidad de Granada (Spain)
E-MAIL: GILSIIS@UGR.ES

ABSTRACT

It is obvious the existence of a powerful connection between quality and productivity in the software projects. Normally, an increase of quality bring to a greater productivity. Maintainability is a very important factor of quality, considering the enormous consumption of resources that is carried out during the maintenance stage. We comment in this article the relation between maintainability and productivity, and when the maintainability result productive and when not.

Keywords: Maintainability, Productivity, Software Quality.

1 Introduction

The productivity, understood like a measurement of the efficiency of the outputs gotten like function of the applied effort, is one of the principal problems expounded in all project software. How could we measure the productivity of the project? How could we utilize the information of previous projects in order to estimate the productivity of a future project? What factors determine the productivity of the project? These are some of the queries that should be considered in all software project.

It is a verified act that, the maintenance stage is the most resources consumer of the entire project. Investigating manifolds coincide telling that, in this stage, it is inverted between the 60% and the 80% of the total cost of the project. Considering this act, if we pretend to act somehow on the productivity of a software software, there will be into account the productivity during the stage of maintenance.

There is a quality factor that determines the productivity during the maintenance, which is known with the name of maintainability. It could be consider like a measurement of the cost and difficulty that supposes the maintenance in a software project [FROS85]

There are certain factors that could be manipulated during the development of a project, with the end of making a product more maintainable. But in the other hand, the act of conferring to the product such characteristics is going to implicate an additional cost. Thus, the work of increment the maintainability of the product, must be properly planned before their realization, determining what is the maintainability characteristics that the product should possess, and what doesn't result of interest.

1.1 Software Quality Economics

When we plan software quality (SQ) activities, we must consider the evaluation of SQ economics, it is to say, costs and benefits of SQ. SQ implies increasing costs: documentation, tools, standards, additional activities, ... In order to justify investments in quality, a return on investment is necessary. The best level of quality isn't the optimal level, in the majority of cases, but an Acceptable Software Quality Range (ASQR), having into account the costs of the quality [ORLA92].

To determine the ASQR, we can consider a cost-quality ratio constituted by the sum of quality costs and quality fault costs (see Fig. 1).

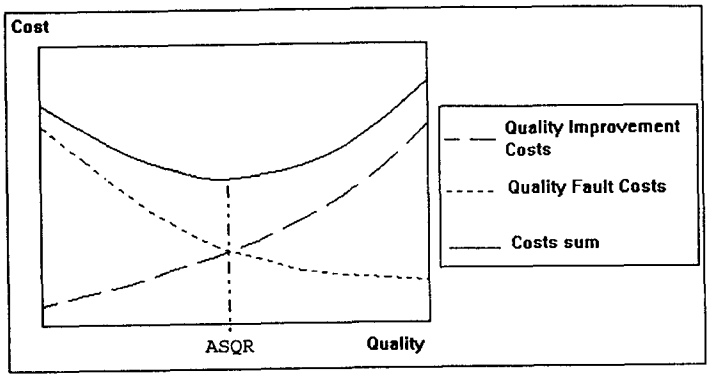

Fig 1. Acceptable Software Quality Range (ASQR) [ORLA92]

2 Productive Maintainability

As other quality factors, maintainability, or maintenance facility, involve costs during the development stage: basically, costs of applying maintainability characteristics to the product, an costs of controlling it. There are three attributes that conform the maintainability: understandability, modifiability and testability. In order to assign, in a productive manner, that attributes to the product, it is necessary to determine the software components that needs this characteristics.

2.1 Economy of Maintainability

As we say previously, not all the components of the system have the same needs of maintainability. The criterion is obvious: the components which are more used during maintenance needs better maintainability than the others. But, how and when can be determined the rate of use that a component will have during maintenance? Well, that is the key question, which we will study immediately.

The objective is to economize efforts in development, applying maintainability characteristics only to those components that need it. As we see in Fig. 2, the investment on maintainability (Maintainability Improvement Cost) must have a return during maintenance (Profit during Maintenance). Otherwise, this is not a productive investment.

We consider a Amortization Period. This is the period in which Maintainability Improvement Cost is equal to Profit during Maintenance. In that moment, the investment is amortized, and after this period, the profit of investment is a real benefit.

2.2 Maintainability levels.

Previous to the development, it will be precise to carry out an estimate of the change that they could surge, with certain

frequency, during the maintenance. So, the most frequent change will demand a greater level of maintainability than the less frequent changes, and then, unnecessary costs will be avoided.

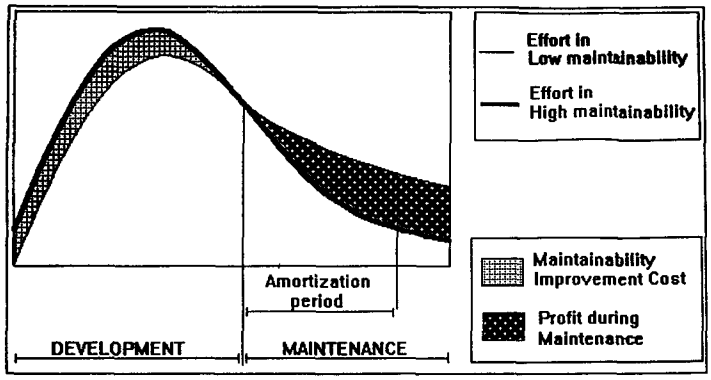

Fig. 2. Productive Maintainability

Abridging this point, we will say that, the **maintainability depend on the type of change.** It will be mission of a collective decisions group, with judgements of experts, (see part 1.1) to determine:

- What is the types of change that could give up during the maintenance.
- Which is the frequency esteemed of each change.

In order to take such decisions, it will be necessary to use the available historic information from similar projects.

The proposed levels are the next:

(Level 3) **High maintainability**
(Level 2) **Middle maintainability**
(Level 1) **Low maintainability**

The maintainability characteristics are of two types:

1. Fixed characteristics: Every software component or module must have some fixed characteristics, so that effort estimates over modules be possible. (i.e., Module size).
2. Variable characteristics: Every software component or module of the same maintainability level, must have the same variable characteristics. (i.e., Number of commentaries in a module).

   Example: We could consider, as an example, the next characteristics:

   Module size: Fixed: Less than 150 lines of code (commentaries excluded)

   No. of commentaries (per module): Variable:
   Level 3 — More than 20 lines of commentaries
   Level 2 — More than 10 lines of commentaries
   Level 1 — No commentaries need.

The best assignment of maintainability levels is that which maximize the difference between the areas "profit during maintenance" and "maintainability improvement cost".

3 Conclusions

Maintainability or ease of maintenance is a factor that needs to be taken into account in the study of the productivity of software projects, given the enormous consumption of resources of the maintenance stage with respect to the remainder of the project.

Several factors, which could be intervened during the first stages of the project, determine the final maintainability of the product. On the other hand, the contribution of maintainability characteristics to a product rebound an additional cost, like any other characteristic of quality. Therefore, one could not give the maintainability characteristics in a indiscriminated form to any product, but rather one must enter to the detail and carry out this contributions module by module; so that useless expenses could be avoid.

4 Futures Research

There are some aspects, related with the proposed model, pendent of investigating:

1. A productivity model based on maintainability
2. A technique to summarize the great amount of data included in the HDB (Historic data base), so that, the result information be more useful to the QCG (Quality Control Group).
3. A technique to help in the MLA (maintainability level assignment) activity, so that, if it is possible, this activity can be automated.

References

ALBR83 Albretch, A.J.; Gaffney, J.E. "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation", IEEE Trans. Software Engineering. Nov.1983.

BANK93 Banker, R.D.; Datar, S.M.; Kemerer, C.F.; Zweig, D. "Software complexity and maintenance costs". Communications of the ACM. Vol 36, No.11. Nov.1993.

BARR94/1 Barranco, M.J.; Granja, J.C. "Control de versiones: Un enfoque práctico". NOVATICA, No.111, Sep-Oct. 1994

BARR94/2 Barranco, M.J.; Granja, J.C. "Estudio gráfico para el análisis de la calidad del software". NOVATICA, No.111, sep/oct. 1994.

BARR94/3 Barranco, M.J.; Granja, J.C. "Estudio de factores de calidad del software partiendo de la ERCU". NOVATICA, No.111, sep/oct. 1994.

BARR95/1 Barranco, M.J.; Granja, J.C. "Proyectos Informáticos". ISBN: 84-600- 8855-3. DL: J-183-94. UNED, 1995.

BARR95/2 Barranco, M.J.; Granja, J.C. "Control of versions: a practical approach oriented to improve the software quality", Proc. of SQM 95, Sevilla, April 1995.

BASI75 Basili, V.R., Turner, A.J. "Iterative enhancement: a practical technique for software development". 1975.

BOEH81 Boehm, B.W. "Software Engineering Economics". 1981.

CONT86 Conte, S.D.; Dunsmore, H.D.; Shen,V.Y. "Software Engineering Metrics and Models". 1986.

FROS85 Frost, D. "Software maintenance and modifiability", Proc.1985 Phoenix Conference on Computers and Communications, Phoenix, Az: 1985

GRAN91 Granja, J.C. "Planificación de Sistemas Informáticos en la gestión empresarial: desarrollo del software y su optimización." Rev. Estudios Empresariales / 4

GRAN92/1 Granja, J.C. "Contribución al estudio de las técnicas de garantía de calidad". Novática. Vol XVIII, No. 99. Sep-Oct. 1992.

GRAN92/2 Granja Alvarez, Juan Carlos. "Aportación a las técnicas de garantía de calidad de software: su incidencia en la planificación", Facultad de Informática de Madrid, U.P.M., 1992.

GRAN92/3 Granja, J.C. "Gestión de la Calidad en Informática", Proc. of USI 92, 1992..

GRAN92/4 Granja, J.C. "Planificación de Sistemas Informáticos en la Gestión Empresarial: Desarrollo del Software y su optimización", Revista de Estudios Empresariales, 1992.
GRAN92/5 Granja, J.C. "Performability", Proc. of IFIP 92. 1992.
GRAN92/6 Granja J.C. "Software Development", Proc. of IFIP 92. 1992.
GRAN92/7 Granja J.C. "Contribucion al estudio de las técnicas de implementación", Proc. of PRODE 92. 1992.
GRAN92/8 Granja J.C. "The vulnerability of information society: Social, Legal and Security aspects", Proc. of IFIP 92. 1992.
GRAN93/1 Granja J.C. "Software Control Project", Proc. of TELETEACHING 93. 1993.
GRAN94/1 Granja, J.C. "Performability","Software Development","Security in Computer Products", Proc. of Fourth European Conference on Software Quality, Basilea, Oct. 1994.
GRAN94/1 Granja, J.C. "Analisis y desarrollo de proyectos informáticos en la gestión empresarial", Revista de Estudios Empresariales, Julio, 1994.
GRAN95/1 Granja, J.C. "Configuración y Dimensionamiento de Equipos Informáticos". ISBN: 84-600-8856-1. DL: J-124-1994. UNED, 1995.
GRAN95/2 Granja, J.C. "Software Development", Proc. of SQM 95, Sevilla, April 1995.
GRAN95/3 Granja, J.C. "Elementos y herramientas en el desarrollo de Sistemas Informáticos: una visión actual de la tecnología CASE", ra-ma, 1995.
GRAN95/4 Granja, J.C. "Técnicas de evaluación de la productividad del Software. Exposición de las posibles tendencias de este campo". Jornadas sobre desarrollo de Sistemas de Información, Madrid, 1995.
GRAN95/5 Granja, J.C. "Nuevas aportaciones a la garantía de calidad del Software" Actas del VI contreso nacional de la calidad. Mayo. 1995.
JONE86 C.Jones. "Software Productivity". McGraw-Hill. 1987.
MCCA76 McCabe, T. "A Software Complexity Measure". IEEE Trans. Software Engineering, vol. 2, Dec. 1976.
ORLA92 Orlandi, E. "Software Quality Economics". 3rd. European Conference on Software Quality. (Madrid - Spain) November 1992.
PICK93 M.M.Pickard, B.D.Carter. "Maintanibility: What Is It And How Do We Measure It?" Software Engineering Notes (ACM SIGSOFT). Vol 18, no. 3. Jul.1993.
WALS77 C.E.Walston. "The Estimation of Software Size and Effort: an Approach based on the Evolution of Software Metrics". 1977.

Book Reviews

Information Modeling – An Object-Oriented Approach

Haim Kilov and James Ross

Information Modeling – An Object-Oriented Approach is written by Haim Kilov and James Ross, and published by Prentice-Hall, 1994. ISBN 0-13-083033-X 252 pp. (plus references and the index) $42.00.

With this book, the authors offer a genuinely useful introduction to the discipline of analysis, attempting to bring rigor to the process of analysis. But when I say 'introduction', I don't mean that the book will serve only the inexperienced reader. As someone with substantial experience, I still found the book beneficial and entertaining. And I believe that it can serve as a valuable refresher on analysis as well as a sound introduction to object-oriented modeling.

In the introduction to their book, the authors write: "This book is about making analysis as disciplined as programming. We show how the analyst may use the same concepts of 'good thinking' as the programmer – abstraction, precise understanding of behavior, and reuse – to end up with a specification that is understandable and formal. The book is not about drawing pictures; it is about formal specification of behavior, at the right level of abstraction, as an approach to system analysis." (p xv)

The authors establish early and return frequently to the essence of analysis: the production of unambiguous models; based on the business knowledge of subject matter experts; with a degree of precision and abstraction appropriate for the intended audience. "Customers are not interested in programs; they are interested in solutions to their information management problems." (p 8) "The job of the analyst is to define and communicate business rules. The singular goal of information modeling is to formulate business rules." (p 19)

The authors make pointed comparisons between software engineering and other branches of engineering. They highlight the ubiquitous problem of complexity, which Brooks identifies as an inherent facet of software: "As an illustration, a single line of code is perhaps eight orders of magnitude smaller than the software system of which it is a part, yet programmers create and maintain both complete software systems and their building blocks, which may be a single line of code. In civil engineering, an atom is perhaps eight orders of magnitude smaller than the structure of which it is part, yet civil engineers do not create or maintain atoms or even molecules . . .." (p 3) The book suggests useful techniques, identifies thinking patterns, and points out traps.

The authors use as their base the notion of Contracts based on assertions. These assertions identify the pre- and post-conditions for operations, and also identify what does not change as the result of an operation – what remains invari-