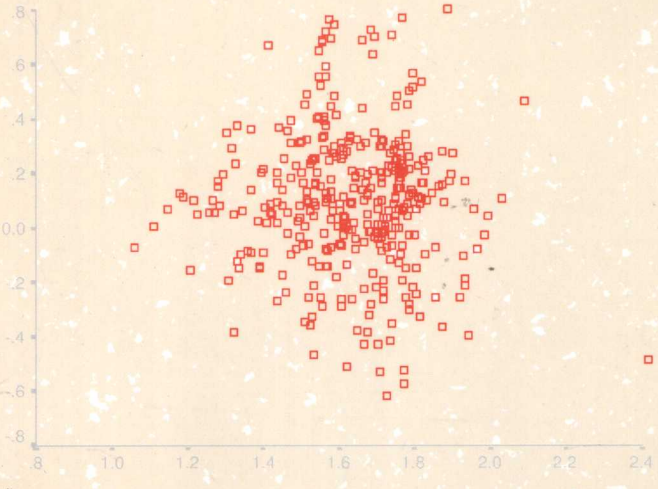
A General Methodology for the Derivation *of* Generalised Software Metrics Models: The Theory and Experiments

Victor Chan



JINAN UNIVERSITY PRESS

A General Methodology for the Derivation *of* Generalised Software Metrics Models: The Theory and Experiments

Victor Chan

JINAN UNIVERSITY PRESS
Guangzhou China

图书在版编目（ CIP ）数据

一种推导广义软件度量模型的方法:理论与实验=A General Methodology for the Derivation of Generalized Software Metrics Models: The Theory and Experiments: 英文 / 陈嘉贤著. —广州：暨南大学出版社，2009. 8

ISBN 978-7-81135-379-2

Ⅰ.一… Ⅱ.陈… Ⅲ. 软件工程—英文 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2009）第 145311 号

出版发行：暨南大学出版社

地　址：中国广州暨南大学

电　话：总编室（8620）85221601

营销部（8620）85225284 85228291 85220693（邮购）

传　真：（8620）85221583（办公室） 85223774（营销部）

邮　编：510630

网　址：http: / /www.jnupress.com http: / /press.jnu.edu.cn

排　版：广州市天河星辰文化发展部照排中心

印　刷：广州桐鑫印刷有限公司

开　本：787mm×1092mm 1/16

印　张：18

字　数：400 千

版　次：2009 年 8 月第 1 版

印　次：2009 年 8 月第 1 次

定　价：36.00 元

Foreword

This book dilates on its underlying research which aimed to innovate a *new* general methodology for deriving *generalised* software metrics models from past empirical software metrics data. These *generalised* software metrics models are to predict the target software metric(s) of any *future* software project from the project's predictor software metric(s) always in a *best-effort*, *best-accuracy* and *best-consistency* manner *whether all, only some or even none of these required predictor software metric(s) is/are available for that future project and/or projects in the past empirical software metrics data.* A software project's predictor software metric(s) indicate(s) or measure(s) the project's relevant software engineering factor(s). This general methodology was illustrated in the research by deriving, from the empirical software metrics data of past software projects sourced in this research, two real-world *generalised* software metrics models with the productivity and work effort measures as the target software metrics respectively. These two real-world *generalised* software metrics models' prediction accuracy and consistency were also assessed in this research. Additionally done was an analysis of the impact of the various aforesaid software engineering factor(s) of software projects on the projects' productivity and work effort. All these works are presented in this book.

This general methodology not only can derive generalised software metrics models capable of dealing with missing software metrics as mentioned above but also offers technical and mathematical solutions to various practical difficulties and imperfections, which afflict most other softeare metrics models. The two real-world *generalised* software metrics models so derived in this research were found to be *superior to the analogous popular existing software metrics models* in term of, say, prediction accuracy and consistency.

Among the multitude of methods employed in this general methodology, the core method is the application of "*expectation-maximisation algorithm for the general location model.*"

Chapters 1 to 3 detail respectively the research's background and objectives, Chapters 4 and 5 the general methodology innovated whereas Chapters 6, 7 and 8 respectively the findings, discussion and conclusion.

If fact, some individual concepts of the foregoing general methodology were published in the following conference papers:

- Chan, V. K. Y. 2004. Software Effort Prediction Models Using Maximum Likelihood Methods Require Multivariate Normality of the Software Metrics Data Sample: Can Such a Sample Be Made Multivariate Normal? *Proceedings, 28th IEEE Annual International Computer Software and Applications Conference (COMPSAC 2004)*
- Chan, V. K. Y. and Wong, W. E. 2005. Optimizing and Simplifying Software Metric Models Constructed Using Maximum Likelihood Methods. *Proceedings, 29th IEEE Annual International Computer Software and Applications Conference (COMPSAC 2005)*
- Chan, V. K. Y. and Wong, W. E. 2007. Outlier Elimination in Construction of Software Metric Models. *Proceedings, 22nd Annual Association for Computing Machinery Symposium on Applied Computing (ACMSAC 2007)*

Nonetheless, the general methodology was not presented in a holistic manner then. In the year 2008, the author publicised the information about the entire general methodology through the title *A General*

Methodology for the Derivation of Generalised Software Metrics Models in the Presence of Missing Data. However, the inclusion of excessive demographic data and experimental data therein overshadowed the 2008 title's intended focus on the theories of the general methodology and the related experimental results, rendering the title's readership circumscribed. On the advice of supporting readers and colleagues, the 2008 title is revised and restructured to alleviate such weakness in presentation, resulting in this book.

Acknowledgements

The author of this book is indebted to the United Kingdom Software Metrics Association (UKSMA), British Computer Society (BCS), International Function Point Users Group (IFPUG), Central Computer and Telecommunications Agency (CCTA) of the United Kingdom, Netherlands Software Metrics Association (NESMA), Mr. Charles R. Symons of the Software Measurement Services, University of Wolverhapmton, University of Bournemouth and many other unnamed organisations and individuals for their provision of invaluable information on the availability of past empirical software metrics data for the research underlying this book. In particular, the International Software Benchmarking Standards Group's (ISBSG's) offer of their Repository Data Disk — Release 6, a sample of such past empirical software metrics data on which most of this research was based, at a concessionary price is cordially appreciated.

Abbreviations

The abbreviations used in this book are defined, in their alphabetic order, as follows:

Abbreviation	Definition
BCS	British Computer Society
CCTA	Central Computer and Telecommunications Agency
DBMS	Database Management System
EM	Expectation-maximisation
FPA	Function Point Analysis
GQM	Goal/Question/Metric
IFPUG	International Function Point Users Group
IQR	Interquartile Range
ISBSG	International Software Benchmarking Standards Group
ISBSG 6 Data	ISBSG's Repository Data Disk — Release 6
IT	Information Technology
LMS	Least Median of Squares
LS	Least Squares
ML	Maximum Likelihood
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
NESMA	Netherlands Software Metrics Association
OLS	Ordinary Least Squares
PC	Microcomputer or Personal Computer
PDR	Project Delivery Rate
RAD	Rapid Application Development
RREF	Reduced Row Echelon Form
SLOC	Source Lines of Code
SPIN	Software Process Improvement Network
UKSMA	United Kingdom Software Metrics Association

Table of Contents

List of Illustrations

List of Tables

1 Introduction

1.1 Software Metrics

Based on the work of Fenton in 1991 and summarised by MacDonell and Gray in 1997, software metrics are measurements relating to a software system (*i.e.* the "product") or to the process of developing the system (*i.e.* the "process"). Examples of "product" software metrics are the size of a system (perhaps in the number of lines of code or the number of screens and reports), the number of defects in a system remaining after testing, some measure of the complexity of a system, *etc.* Examples of "process" software metrics may include the number of developers involved in the software project to develop the system, the work effort required for various stages of the project, and the experience of the developers, *etc.* Note that the number of screens or reports is usually classified as a functionality-based measure.

Traditionally, such software metrics were used as part of a formally specified model, such as "function point analysis" (Garmus and Herron 1995 and §1.4.1), which could be calibrated to a specific organisation and/or environment. Alternatively, they were used as variables in other models that established relationships between them for different software systems or projects. Regression equations are typical examples of these models. These relationships relate a target software metric (to be defined in §2.1) or target metric, in short, with one or more predictor software metric(s) (to be defined in §2.1) or predictor metric(s), in short. For example, the work effort (*e.g.* in the number of programmer hours) required for testing a particular system or a series of its modules might be the dependent variable in a regression equation with a functionality-based measure of the size of the concerned system, the system's complexity and the developers' experience as the independent variables. All such models of this type are known as *software metrics models*. Note that "dependent variable" is a term in statistical regression corresponding to the target metric here whilst "independent variable" is a term in statistical regression corresponding to the predictor metrics here.

1.2 Software Metrics Models

Means to derive software metrics models relating various software metrics have been sought since the advent of the study of software metrics (Zhou and Leung 2006, Côté *et al.* 1988, Shepperd 1988, Gremillion 1984 and DeMarco 1982). Typically, these models relate the work effort required for software projects to develop different software systems with a functionality-based measure of the sizes of the systems, their complexities and their developers' experience. Once derived, software metrics models can be used to predict one or more target metrics, given the predictor metrics. The aforesaid work effort is a typical example of target metrics while typical examples of predictor metrics may include the aforesaid functionality-based measure of the sizes of the systems, their complexities and their developers' experience. In his inspired assertion of software science (Halstead 1977) which was unfortunately seen subsequently as somewhat flawed, Halstead, one of the founders of software measurement, included an equation to predict the work effort as the target metric for program development. The equation was based on the fundamental algorithm size which itself composed several predictor metrics. Software metrics models have become a pivotal mechanism to control aspects of the "process" of the project to develop a system. Examples of these aspects may include the work effort, resources, costs required to develop the system, *etc.* Software metrics models should have real-world practical values and uses to software projects.

On the subject of software metrics, MacDonell and Gray 1996 presented an inspiring and enlightening comment

> "It [it] is one thing to measure and record data of interest; it is another to analyse and interpret that data in a valid and reliable manner.[,]"

which implicitly indicates that to derive a software metrics model, there are two processes:

1. The selection (or shortlisting as referred to as in the general methodology innovated in this research) of software metrics for the software metrics model, including presumably the target metric(s) and predictor metric(s) that are expected to impinge on the target metric(s).
2. The construction of the software metrics model relating the target metric(s) with the predictor metric(s).

Also, it is noteworthy that the construction of most software metrics models is in some way based on the empirical software metrics data of past software projects. Such data usually comprises the target metric(s) and predictor metric(s) for each of these past projects. In essence, the construction of the models becomes the establishment of the relationship between the target and predictor metrics as "exemplified" by these past software projects.

Generalised software metrics models (or *generalised* models, in short), repeatedly mentioned in this book, refer to software metrics models that are to predict the target metric(s) for any *future* software project (or *future* project, in short) from the predictor metric(s) for the project always in a *best-effort, best-accuracy* and *best-consistency* manner whether all, only some or even none of these predictor metric(s) required for the model is/are available for that *future* project and any past projects used for the construction of the model. In short, a *generalised* model is a software metrics model characterised by its missing data processing capability.

1.3 Importance of Software Metrics Models

In view of the large expenditures made by many companies and organisations on the development of software systems, even small increases in prediction accuracy and consistency with respect to productivity, work effort, speed of project delivery, quality, *etc.* are likely to be worthwhile and have considerable value to any company or organisation involved in software engineering/development. Underestimating work effort and thus costs can lead to an acceptance of software projects that do not provide sufficient returns or that overrun budgets and schedules. Overestimating work effort and thus costs can lead to sound projects being rejected, and can lead to gaps between one project ending and another starting. A recent example of work effort underestimation in the United Kingdom is the Legal Court System which was originally estimated at a cost of £148 million by the contractor but turned out to cost nearly £500 million.

Nevertheless, owing to the importance of software metrics models, once a model has been derived, it is essential that the limitations of the methodologies used to derive the model be understood in order to ensure that the model is only used within its limitations. For example, extrapolations outside the range of software metrics data used for the derivation of a software metrics model could lead to significant error.

1.4 Existing Software Metrics Models

§1.4.1 to §1.4.8 outline some existing software metrics models. In particular, the comparative advantage(s) and disadvantage(s) of each of them are presented. Save for function point analysis, all the software metrics models here define only process 2 of §1.2 while process 1 is left to the particular practitioners of the models. See Gray and MacDonell 1997 and MacDonell and Gray (1997 and 1996) for

details. In the papers by MacDonell and Gray (1997 and 1996), there is in-depth quantitative comparison between the main existing software metrics models in terms of their empirical performance in prediction accuracy and consistency whilst in the 1997 paper by Gray and MacDonell, there is a criteria-based comparison between these models.

1.4.1 Function Point Analysis

1.4.1.1 Overview

Proposed by Albrecht (1979), function point analysis (FPA) specifies not only the construction of the software metrics model (*i.e.* process 2 of §1.2) but also the set of selected software metrics in its model (*i.e.* process 1 of §1.2). The construction of FPA's software metrics model is heuristic in nature and intends to relate the predicted work effort (being a "process" metric and as the target metric) for a software project to develop or maintain a system with a set of "product" metrics (as predictor metrics) of that system. The systems involved in FPA are typically transaction processing systems. A major "product" metric in FPA is the *unadjusted function points* count which measures the "information processing size" of the system. This *unadjusted function points* count is adjusted or, more exactly, multiplied by the *technical complexity adjustment* (TCA and alternatively known as the *value adjustment factor*) of the system to give the *adjusted function points* count which is plugged directly into empirical equations to predict the work effort. These empirical equations are usually calibrated for particular organisations. The TCA consolidates 14 "product" metrics other than the *unadjusted function points* count and each of them measures a "general application characteristic" of the system in question. Other than the original Albrecht standard, there are now several variant FPA standards, including various sub-versions of the International Function Point Users Group (IFPUG) standard and Mark II FPA which is a revised version of Albrecht FPA and was developed by Charles R. Symons (Symons 1991 and 1988 and Drummond 1992). In recent years, there have still been quite some review and revision of various variants of FPA (Kralj et al. 2005.and Hastings and Sajeev 2001).

Adhering to the IFPUG standards but ignoring the subtle difference between the various sub-versions, FPA could be detailed as follows. Each component of a system (*e.g.* an input screen, a report, a database table, *etc.*) is subsumed under one of the five component types, namely,

- internal logical file,
- external interface file,
- external input,
- external output and
- external inquiry.

Each such component of a particular component type is further classified as "simple," "average" or "complex" according to Table 1.1.

Internal Logical File				External Interface File			
Record Element Types	Data Element Types			Record Element Types	Data Element Types		
	1 – 19	20 – 50	⩾ 51		1 – 19	20 – 50	⩾ 51
1	Simple	Simple	Average	1	Simple	Simple	Average
2 – 5	Simple	Average	Complex	2 – 5	Simple	Average	Complex
⩾ 6	Average	Complex	Complex	⩾ 6	Average	Complex	Complex
External Input				**External Output**			
File Types Referenced	Data Element Types			File Types Referenced	Data Element Types		
	1 – 4	5 – 15	⩾ 16		1– 5	6 – 19	⩾ 20
0 – 1	Simple	Simple	Average	0 – 1	Simple	Simple	Average
2	Simple	Average	Complex	2 – 3	Simple	Average	Complex
⩾3	Average	Complex	Complex	⩾ 4	Average	Complex	Complex
External Inquiry							
File Types Referenced	Data Element Types						
	1 – 4	5 – 15	⩾ 16				
0 – 1	Simple	Simple	Average				
2	Simple	Average	Complex				
⩾ 3	Average	Complex	Complex				

Table 1.1 Classification of software system components of different component types as "simple," "average" or "complex."

Then an *unadjusted function points* count is awarded to each such "simple," "average" or "complex" software system components as in Table 1.2.

Component Type	A Component Classified as		
	Simple	Average	Complex
Internal Logical File	7	10	15
External Interface File	5	7	10
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6

Table 1.2 The *unadjusted function points* count awarded to a software system component of a particular component type and classification as "simple," "average" or "complex."

The sum of the *unadjusted function points* counts of all the system components becomes the *unadjusted function points* count of the system itself. Also, for the system, a *degree of influence* is assigned to each of its 14 "general application characteristics" as enumerated below:

- data communications,
- distributed functions,
- performance,
- heavily used configuration,
- transaction rate,
- on-line data entry,