OPEN SOURCE SOFTWARE MATURITY MODEL BASED ON LINEAR REGRESSION AND BAYESIAN ANALYSIS

A Dissertation

by

DONGMIN ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2007

Major Subject: Computer Science

OPEN SOURCE SOFTWARE MATURITY MODEL BASED ON LINEAR REGRESSION AND BAYESIAN ANALYSIS

A Dissertation

by

DONGMIN ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Dick B. Simmons
Committee Members,	William M. Lively
	Udo Pooch
	Ergun Akleman
Head of Department,	Valerie E. Taylor

August 2007

Major Subject: Computer Science

ABSTRACT

Open Source Software Maturity Model Based on Linear Regression and Bayesian Analysis. (August 2007)

Dongmin Zhang, B.S., Tsinghua University, P.R. China;

M.S., Tsinghua University, P.R. China;

M.S., University of Nebraska-Lincoln

Chair of Advisory Committee: Dr. Dick B. Simmons

Open Source Software (OSS) is widely used and is becoming a significant and irreplaceable part of the software engineering community. Today a huge number of OSS exist. This becomes a problem if one needs to choose from such a large pool of OSS candidates in the same category. An OSS maturity model that facilitates the software assessment and helps users to make a decision is needed. A few maturity models have been proposed in the past. However, the parameters in the model are assigned not based on experimental data but on human experiences, feelings and judgments. These models are subjective and can provide only limited guidance for the users at the best.

This dissertation has proposed a quantitative and objective model which is built from the statistical perspective. In this model, seven metrics are chosen as criteria for OSS evaluation. A linear multiple-regression model is created to assign a final score based on these seven metrics. This final score provides a convenient and objective way for the users to make a decision. The coefficients in the linear multiple-regression model are calculated from 43 OSS. From the statistical perspective, these coefficients are

considered random variables. The joint distribution of the coefficients is discussed based on Bayesian statistics. More importantly, an updating rule is established through Bayesian analysis to improve the joint distribution, and thus the objectivity of the coefficients in the linear multiple-regression model, according to new incoming data. The updating rule provides the model the ability to learn and improve itself continually.

To Li and my family,

for their love,

support and encouragement

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Dr. Dick Simmons who accepted me as his student when I was in the most difficult time on my way of pursuing a Ph.D. He gave me invaluable guidance, inspiration, and tremendous support and encouragement in my graduate study and research at Texas A&M University. Without him, this dissertation would not have been possible. His knowledge, experience, and wisdom guided me throughout my research and benefited me, not only during the time I was working on my dissertation, but also for the rest of my life.

I want to express deep appreciation and gratitude to Dr. William Lively for his help and comments in the area of Software Engineering and his encouragement and support for this research. I would also like to thank Dr. Udo Pooch for his support and advice that helped me during the critical phase of my study. I am specially grateful to Dr. Ergun Akleman for his class and the suggestions he gave for my research.

I am very pleased to have had the chance to work with other members in Dr. Simmons's research group: Guobin He and Norman Ma. I am grateful for their helpful discussions and the pleasant times we had together.

Finally, I would like to thank my husband, Li and my family. They gave me endless love and support throughout all these years. Without them, I would not have been able to reach this goal.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Software Maturity model is a process model used to assess the level of maturity of an organization's software development process. It helps the developers locate the weak points in their software process and helps them to improve it.

I.1 Traditional Software Maturity Model

In 1988 and 1989, Watts Humphrey, from Software Engineering Institute, proposed a software process maturity framework [1][2]. His maturity framework includes 5 process levels:

1. "Initial" level. It is also known as Ad Hoc, or even chaotic. At this level, software management tools are not uniformly employed in the software process. The management team does not realize the benefits of software engineering methods and technologies such as project planning, code design and cost estimation.
2. "Repeatable" level. At this level, the organizations follow rigorous management procedures: commitment control, cost and schedule control.

This dissertation follows the style of *IEEE Transactions on Software Engineering*.

3. “Defined” level. At this level, the organizations have well-defined software process, including activities, plans, resources, constrains and objectives.
4. “Managed” level. At this level, software process measurement is introduced and applied to each part of the process, and the measurement yields statistical data and analysis to help the management of the software process.
5. “Optimizing” level. Based on the result of measurement and analysis in Managed level, improvement can be applied to the software process.

In the following years, based on the 5-level maturity model, Software Engineering Institute (SEI) presented their Capability Maturity Model (CMM) [3] that has since become a widely used model for assessing and improving software process. This model is based on the traditional software development process that consists of planning, cost estimation, scheduling, resources allocation, quality control, process management, etc. This model will be described in more details in Chapter II.

I.2 Open Source Software

Open Source Software refers to the software whose source code is available to public under certain terms and users have the right to read, modify, and derive from the source code. Also, users can distribute the original and modified version of the software in source code format and/or in compiled format under the same license as the original one.

According to the Open Source Definition provided by Open Source Initiative, Open Source Software must comply with the following 9 metrics.

1. Allow free redistribution. There is no restriction on selling and free offering of the software. No loyalty and fee should be charged for redistribution.
2. Open source code. Source code must be publicly available and can be obtained free of charge. No fees should be charged on redistribution of the source code.
3. Allow redistribution of derived work. Modifications of the source code are allowed. And the modified version can be redistributed under the same license.
4. Integrity of Author's source code. In order to keep a record of who is responsible for a particular part of the code, the license must allow distribution of patches instead of modified version of the program.
5. No discrimination against certain people or a group of people. Every person is eligible to contribute to the open source software. There should be no restriction on who can and who cannot get involved in the software development.
6. No discrimination against certain field. The program should be used in all fields including business and research.
7. No need for additional license. When the program is distributed, whoever receives the program has all the right listed in the license without any additional terms.
8. License is attached to program other than product. Terms are applied to the program. It does not matter if the program is part of a product or not.

9. License is applied to the program only. The license can be applied to the particular program only. It should have no restriction on other programs.
10. License must not prefer some technology than others. The license must have no restriction on how the program will be obtained and how it will be used.

Open Source Software has these special features. They are quite different from those of traditional software. Open Source Software is usually contributed by individual contributors. These contributors usually do not stay together physically. Sometimes they may live in different countries or even continents. Open Source Software process has no scheduling, resource allocation and management involved. No managers assign tasks and deadlines to the developers. The individual contributors volunteer their time and efforts, and they cooperate in very loosely manner. No hierarchical leadership or formal management structure is involved. According to a survey study [4], around 60 percent of the Open Source Software contributors and developers are not being paid by any company and most of them are motivated by their own personal interest.

I.3 Motivation

The development process of Open Source Software is very different from traditional software development process in many ways. It is hard to apply the traditional software maturity model directly on Open Source Software development process. We need to study and build an Open Source Software maturity model for the following reasons:

1. The need of an individual user for an evaluation model. As mentioned above, Open Source Software is getting more and more attentions and market shares today. However, there are so many OSS out there and there seems to be no way to tell which one is the best. As a software engineer, the author had some personal experiences of choosing from a large pool of OSS, which is a time-consuming, challenging and confusing process. For example, when several desirable candidates with similar functionalities are chosen, it is hard to determine which one to use. Since there are no user reviews available, the user has to spend a large amount of time on each of the candidates in order to evaluate them before the final decision is made. On the other hand, if there were some model that could help evaluate all the candidates, the user would have been able to get rid of those relatively immature ones quickly and get focused on the promising ones. Thus the user would save a lot time and get a better chance to choose a good one and save on cost. It is especially dangerous if the users pick a wrong one and make it the foundation for software development and after some relatively long period realize that they have chosen the wrong one in the first place. Users, especially those inexperienced ones, need guidelines or a good OSS maturity model to help them make the decision.
2. Certain needs arise recently. Many government departments and organizations have recently expressed their need for OSS maturity model. For example, California government has started to consider Open Source Software alternatives, and they have realized the importance of maturity model for the OSS selection

process [5]. Government of Canada has also voiced its needs for Open Source Software. All these governments and organizations need a standard procedure to help them choose among OSS with similar functionalities [6].

3. OSS maturity model has become a research frontier in the software engineering community. Part of open source community has switched its focus to OSS maturity model recently. Carnegie Mellon West Center for Open Source Investigation, cooperating with Intel, O'Reilly CodeZoo, and SpikeSource, proposed Business Readiness Rating as a standard model for rating open source software [7]. David A. Wheeler thoroughly explained why we need to build OSS maturity model and proposed a model [8]. Navica/Golden proposed Open Source Maturity Model [9]. There are many other researcher groups focusing on this topic [10][11][12][13][14][15]. The existing work is admirable but needs improvements before it can be employed in the real world, which will be explained in more details in the next section.

For all the reasons above, the need for developing a better OSS maturity model is truly urgent. The goal of this research project is to propose a user friendly and objective OSS maturity model. This model can be used to compare OSS, provide users with a quantitative index, assist users to pick up the more mature one, and help them choose between different OSS packages. Certain objective criteria will be defined and a regression model will be developed to compute the scores for OSS. With this maturity model, users need to collect data for each criterion, feed the data into the model, and get the maturity score of their OSS candidates.

I.4 Organization of the Dissertation

This dissertation is organized as the following. Chapter II gives a brief summary of existing software maturity model, including the maturity model for proprietary software and Open Source Software, and states current problems in the existing models. Chapter III describes the metrics used in the proposed model in this dissertation in details, and the reasons to choose these metrics. Chapter IV presents the linear multiple regression model, validation of assumptions, and verification of the regression model. Chapter V explains Bayesian analysis and the model updating rules based on Bayesian analysis, which is the statistical foundation of the research in this dissertation. Finally, Chapter VI concludes the work and gives possible future research directions.

CHAPTER II

EXISTING SOFTWARE MATURITY MODELS

Existing software maturity models are valuable literatures for developing new maturity model. The existing models show the key structure and elements of the maturity model. Studying the methods, techniques and procedures of building existing models provides inspiration for new models. We first look at the most widely adopted Software Maturity Model for proprietary software. Then the common techniques used in the area of software modeling are discussed. Starting from Section 3, the existing Open Source Software Maturity Models are introduced. Each of them is explained in detail and the techniques they use are summarized. The similarities of the models are also studied.

II.1 Software Maturity Model for Proprietary Software

The most popular software maturity model for proprietary software is Capability Maturity Model (CMM) and Capability Maturity Model Integration (CMMI). CMM was developed by Software Engineering Institute (SEI) at Carnegie Mellon University in the 1980s. It is adopted by a lot of companies, organizations and government projects.

Capability Maturity Model is a software process model that can be used to assess the Maturity Level of an organization. It is a tool that can be used for quality

management and is of great help for the organization to improve its process. A process is the way in which people utilize certain resources, follow some procedures and apply a set of methods to achieve their objectives and produce the final desired results.

CMM includes Maturity Levels, Key Practice Areas, Goals, Common Features, and Key Practices. The layered Maturity Levels are well-defined stages covering the evolution from a basic and initial stage to mature software organization. CMM includes five Maturity Levels, which are:

1. Level 1 – Initial
2. Level 2 – Repeatable
3. Level 3 – Defined
4. Level 4 – Managed
5. Level 5 – Optimizing

At level 1, initial, the processes are usually designed for a single project. There are no certain standards to follow. The project management is in chaos. The success in the past is not repeatable in the future. Very experienced and capable people are critical to the success of the organization at level 1. Usually, it is easy for the organizations at Maturity Level 1 to miss the deadline and consume more resources than their initial budget.

At level 2, repeatable, the processes are repeatable for all the projects. The success is also repeatable in the future. The very basic project management is often used. A development plan is designed before implementation and the plan is closely followed during the entire process of the software development. The organizations use scheduling

tools and a resource tracking system to help them stay on time and deliver software products without spending beyond their budget. At this level, the organization can repeat their success for the previously accomplished tasks. For new tasks the organization may repeat their success but under high risks of overflowing budget and late delivery of products.

At level 3, defined, the process is well defined. The organization defined standard processes that all of the projects must follow. The standard process may be modified in each project according to its special requirements and the modification guide.

Unlike level 2, at which the processes are quite different from project to project, at level 3, all projects define their descriptions and procedures based on the same set of standard processes. They are disciplined and vary only based on individual objectives and needs. At this level, the organization is qualitative measurable.

At level 4, managed, quantitative measurement and statistical methods are deployed. Quantitative information is fed to the project management. This provides progress tracking and resource control. At this level, the process in an organization is quantitatively measured and controlled. It is therefore quantitative measurable.

At level 5, optimizing, the organization continuously improves the process. Organizations use new technologies to help them optimize the process. Quantitative improvement objectives are defined and the organization focuses on continuous improvements and optimizing the process to achieve its business goals.

Figure 2.1 shows the 5 levels and the way to improve from one level to the upper level.

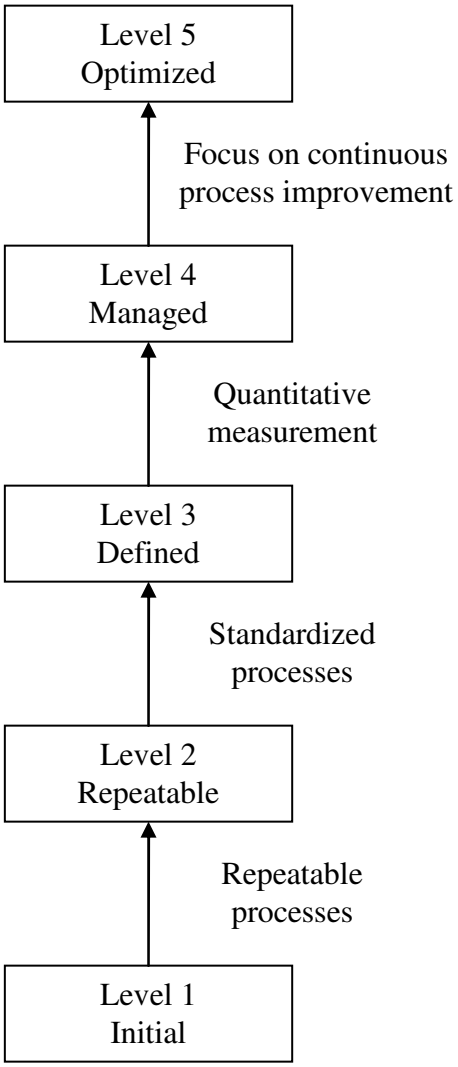


Figure 2.1. Capability Maturity Model five maturity levels evolution

Besides the Maturity Levels described above, Capability Maturity Model defines Key Process Areas (KPA) [16] that consists of a group of activities. These activities are identified to address the issues for an organization at a certain Maturity Level and ensure that the right personnel, resources and other participants get involved. If the

organizations perform these activities, they are guaranteed to achieve the goal that is important to improve the organization's maturity.

CMM defined a set of Key Process Areas for each Maturity Level as shown in Table 2.1.

CMM Goals define a set of status that the organization has to achieve in each Key Process Area, thus to identify the improvement and status of the organization at each Maturity Level. For example, the goals for Software project planning include

- Software estimates are well defined and documented for planning and tracking.
- Appropriate personnel or groups agree with the planned activities and commitment.
- The planned activities and commitments are documented.

CMM Common Features are used to organize the practices in each Key Process Area and indicate if the practices are effective and repeatable in a lasting way. The Common Features include

- Commitment to perform.
- Ability to perform.
- Activities performed.
- Measurement and analysis.
- Verifying implementation.

Table 2.1. CMM key process areas for each maturity level

Maturity Level	Key Process Areas
Improve from Level 1 Initial to Level 2 Repeatable	• Software configuration management • Software quality assurance • Software subcontract management • Software project tracking and oversight • Software project planning • Requirements management
Improve from Level 2 Repeatable to Level 3 Defined	• Peer reviews • Intergroup coordination • Software product engineering • Integrated software management • Training program • Organization process definition • Organization process focus
Improve from Level 3 Defined to Level 4 Managed	• Software quality management • Quantitative process management
Improve from Level 4 Managed to Level 5 Optimizing	• Process change management • Technology change management • Defect prevention

The commitment to perform defines the set of actions the organization must execute to guarantee the planned activities will happen at the certain stage of the process. It may include sub-features, such as policies, responsibilities, and senior management sponsorship.

"Ability to perform" describes the conditions that the organization must meet in order to ensure the process be established and completed. The typical sub-features include resources, organization structure, delegation, training, and orientation.

"Activities performed" explains what roles an appropriate personnel or group should play and what procedures they should follow to implement the Key Process Area. Usually it includes sub-features such as developing plans and procedures, executing the work, tracking the progress, and taking necessary actions to help keeping the process on track.

"Measurement and analysis" includes the quantitative measurement of the process and analysis of the quantitative measurement data. Sub-features include measuring the activities performed and analyzing the measurement results.

"Verifying implementation" describes the procedures and actions that need to be done to make sure the activities comply with the planed and documented process. The typical sub-features are the reviews and tests by senior management members, program managers and the testing and quality assurance team.

CMM Key Practices include the infrastructures and practice that are the most important for effective implementation and institutionalization for each key area. For example, estimating the size of the product is a key practice.

Capability Maturity Model Integration (CMMI) [17][18][19][20] is upgrade and replacement for Capability Maturity Model. CMMI is also developed by Software Engineering Institute (SEI) at Carnegie Mellon University and was first released in 2002. Following the transition in software life cycle process, from Waterfall model to Spiral Software Life Cycle model, to Natural Milestone Life Cycle model, and to Extreme Programming and Agile Programming model, SEI developed CMMI which integrated new models, new process areas, modern key practices, and more implementation goals.

For example, the following are some of the new Key Process Areas that are added to each Maturity Level:

- Newly added Key Process Areas at Level 2:
    - Measurement and analysis
- Newly added Key Process Areas at Level 3:
    - Requirements development
    - Technical solution
    - Product integration
    - Verification
    - Validation
    - Risk management
    - Decision analysis and resolution

II.2 Multiple Regression and Bayesian Analysis Used in COCOMO II

COCOMO II is a cost estimate model and it is one of the most popular Software Engineering models used today. It gives an estimate of the manpower needed to develop a software project. COCOMO II is an update of COCOMO, which is the acronym for "COstructive COst MOdel". COCOMO was introduced by Barry Boehm in 1981 [21] based on a study of 60 software projects developed in a company called TRW Incorporated. The software projects studied to build the COCOMO model have sizes ranged from 2000 to 10,000 lines of source codes and include a few different programming languages from assembly to PL/I.

The COCOMO model includes three forms:

- Basic Model
- Intermediate Model
- Detailed Model

The basic COCOMO model includes the following three equations to estimate the costs

$$E = a_b (KLOC)^{b_b} \tag{2.1}$$

$$D = c_b E^{d_b} \tag{2.2}$$

$$P = E / D \tag{2.3}$$

where E is the estimated efforts in man-month,

$KLOC$ is the size of the software in thousand lines of source code,

D is the estimated development time in months,

P is the estimated number of programmers needed to deliver the software,

a_b, b_b, c_b, d_b are the coefficients and are different for different types of software projects in basic COCOMO form, as shown in Table 2.2.

Table 2.2. Coefficients in COCOMO model

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

COCOMO II [22] was released in mid of 1990s and it is the update of COCOMO model. COCOMO II published in 1997 is based on multiple regression approach. It has five "Scale Factors" and seventeen "Effort Multipliers" [23]. Its mathematical form is expressed as Equation (2.4) [24]

$$E = A \times [Size]^{1.01+\sum_{i=1}^{5} SF_i} \times \prod_{i=1}^{17} EM_i \tag{2.4}$$

Where E is the estimated effort,

A is a constant,

Size is the size of the software, which is measured by thousand lines of source code,

SF is Scale Factor,

EM is the effort multiplier cost driver.

"Thousand lines of source code" is used to represent the size of the project. Other researches suggested other metrics, such as Chuck metric introduced by Simmons et al.[25], function points proposed by International Function Point Users Group [26], and object points suggested by Banker et al. [27].

If we take logarithms on both sides of Equation (2.4), we get a linear equation, as shown in Equation (2.5). The coefficients associated with each term are determined by the multiple regression method based on 83 data points [24] collected from commercial software.

$$\begin{aligned}\lg(E) = \beta_0 + \beta_1 \cdot 1.01 \cdot \lg(Size) + \beta_2 \cdot SF_1 \cdot \lg(Size) + \beta_3 \cdot SF_2 \cdot \lg(Size) + ... \\ + \beta_6 \cdot SF_5 \cdot \lg(Size) + \beta_7 \cdot \lg(EM_1) + ... + \beta_{23} \cdot \lg(EM_{17})\end{aligned} \quad (2.5)$$

Some counter intuitive results are found in COCOMO II. Therefore Chulani et al. proposed to use Bayesian Analysis to calibrate COCOMO II [24][28].

This marks the inspiration for the statistical methods used in this dissertation. As explained in details in Chapter IV and Chapter V, a multiple regression model is built as the Open Source Software Maturity Model and Bayesian Analysis [29][30] is applied to update the model.

II.3 Existing Software Maturity Model for Open Source Software

There are a variety of existing models. Among them are the three major ones, which are Open Source Software Maturity Models, Navica/Golden Open Source Maturity Model(OSMM), CapGemini Open Source Maturity Model, and Business

Readiness Rating. These maturity models have some common criteria and share some common methods. Following is the summary for each of these models.

Navica/Golden Open Source Maturity Model (OSMM)

Navica/Golden Open Source Maturity Model was proposed by Bernard Golden [9]. It was considered to be the pioneer of Open Source Maturity Model (OSMM).

Golden provided a template for the maturity assessment, which consists of three phases. Phase One: Identify and assess important metrics for Open Source Software maturity. Phase Two: Define the weighting factor for each metric picked up in Phase One. Phase Three: Compute the final maturity score by summing weighted metrics' scores.

During phase one, six elements are identified as the key metrics: "Product Software", "Support", "Documentation", "Training", "Product Integrations", and "Professional Services". Phase One consists of 4 steps. The first step is that users define their requirements and identify the key criteria for their needs. This step is very important for building the maturity model. The second step is to locate the resources either from within the organization or from the OSS community. This might be challenging compared with the traditional software development process. Then users need to evaluate the maturity of each metric for the OSS. Finally they assign a number as the score for each of the key metrics with experts' help.

Since all criteria are not equally important, in the second phase, users have to give each criterion a weighting factor with experts' help based on their particular needs. The more important the metric is, the heavier the weighting factor should be.

With the scores for key criteria and their weighting factors available, the last phase is to put them together and calculate the final numerical maturity score for the OSS.

OSMM provides a template with their suggested weighting factor and potential score for each of the metrics. The multiplications of the potential scores with the weighting factors sum up to 100. Users need to fill out the actual score for each metric and adjust the weighting factors with experts' help based on their specific requirements.

CapGemini Open Source Maturity Model

CapGemini Open Source Maturity Model was developed to help users to determine if OSS fits the requirements of an organization [31].

This model defines 12 metrics to measure the successfulness of OSS and group them into four categories:

- "Product"
- "Integration"
- "Use"
- "Acceptance"

Their categories are called "Product Indicator".

"Product" group represents the internal quality of the open source project. It includes 5 important metrics:

- "Age". It is the time period during which the Open Source Software keeps active. The longer the age the less chance that the software will be abandoned suddenly.
- "Selling points". These are the features that are not provided by existing software and address the key weakness of existing software, or they are features that are provided with much better quality than their counterparts in existing software. Open Source Software with very good selling points could quickly gain market share.
- "Developer community". The larger the number of people involved in software development, the better chance that the software continues to grow and get improvement. A community with actively involved developers keeps the open source software active.
- "Human hierarchies". A project fully controlled by a single person is hard to grow. Hierarchical management makes the development, code review, testing and support easier to control. Hierarchical management is a wise choice for good software development.
- "Licensing". Very restrictive license may limit possible future use or distribution of the derived work. Choosing the Open Source Software with the appropriate license is very important.

"Integration" group focuses on the features that make the entire open source project or part of it easily integrated with other projects. It includes

- "Collaboration with other products". First, a product should have high quality and be easy to use. The second step is that the product should be easily used in cooperation with other software. PAM (Plugable Authentication Module) is a good example that used by a lot of other applications such as rsh, rlogin and rcp. How well and easily an Open Source Software can be integrated into other software decides if it can be chosen and adopted as a component in the software an organization is developing.
- "Modularity". After an open source product becomes a successful and widely used software project, some people may want to use only part of it based on their needs. If the source code is designed and developed in modules according to different functionalities and each module has clear and well defined interface, each module can be then easily adopted by other projects when only that part of the software is desired. Modularity makes the open source software become more useful as a integrated product, as well as individual functional components. For example, XFree86 was originally designed to require the details of video card. In order to make the video card work with XFree86, manufactures need to open their video card source code which most manufactures are not willing to give up. XFree86 then improved its design and become an X server that can take binaries of the video card.

After that, XFree86 can work with significantly more video cards and more successful software.

- "Standards". One service or protocol usually has many standards established by different companies or organizations. For example, there are three ways to connect to database, direct connection which is supported by only a few database, using ODBC which is a standard in Unix/Linux and Windows, and OLEDB which is a standard in Windows. Certainly, the one using ODBC is preferred. In order to ensure that the software can easily adopt or be adopted by other software, it is very important to comply with most commonly used standards.

"Use" group measures how many ways the software provide to support new users and existing users. It includes 2 metrics:

- "Ease of deployment". When an open source project takes more and more market share, more people become interested in the whole product or part of it. In this case, the documentation that explains how to deploy the whole application and how to interface and contribute to each individual component is critical. Such documentation saves new users a lot of time for seeking around and enables them to get on the right track more easily.
- "Support". When users have problems using the software or request new features, they need support from the developers and the user communities.

"Acceptance" group indicates if the users are satisfied with the open source software and if the software is popular. It includes 2 metrics:

- "User community". Some Open Source Software have a big and active user community, while others may have relatively small group of users. A big and active user community may have important impact on the future of the Open Source Software. Open Watcom C++, for instance, is the result of strong support of the user community.
- "Market Penetration". Market share is an important indicator. Market share represents the percentage of the users who adopt a particular software project. A higher percentage indicates a higher level of maturity.

CapGemini Open Source Maturity Model has provided an example score table for each of the above indicators. The suggested scoring scheme is based on their extensive experience from reviewing and evaluating Open Source projects.

This model also defined 15 metrics to represent the environmental aspects and the current and future demands of the user, such as usability, interfacing, performance, reliability, security and so on.

When users need someone to help them choose an Open Source project, CapGemini consultants with extensive Open Source Software assessment experiences will then help them to assign a score for each of the metrics defined earlier, assign a priority to each metric, and compute the final weighted score for Open Source Software.

Business Readiness Rating

Business Readiness Rating was developed by Anthony Wasserman, Murugan Pal, and Christopher Chan, from OpenBRR.org in a published whitepaper, in which they proposed an "Open and Standard Model for Software Assessment" [7].

They believe that a good model should include both good and bad characteristics of a software product, and should not be biased. It should be easy to understand and easy to use. Also, the model should capture the new changes of Open Source software, and reflect the changes. Furthermore the model should generate a consistent score for software even from different categories.

There are four phases when using this model to assess Open Source project.

- Phase one: "quick assessment". In this phase, user needs to identify their metrics, measure the Open Source software and filter out the ones that don't meet their basic requirements.
- Phase two: "Target usage assessment". Business readiness model defined 12 categories and each category includes a few metrics. In this second phase, users need to pick 7 or fewer more important categories and give percentage category weights that sum up to 100%. Within each category, users rank metrics and assign a percentage weighting factors that sum up to 100% according to their importance.
- Phase three: "Data collection and processing". Data for each metric in each category are collected, assessed and the weighted scores are computed.

- Phase four: “Data translation”. The numeric maturity score is computed.

This model also provides commonly used metrics and their scoring.

II.4 Other Related Work

David Wheeler proposed a procedure to evaluate Open Source Software in [8]. His procedure consists of four phases, namely Identify, Read Reviews, Compare, and Analyze. In the “Compare” phase, he proposed 13 metrics that users should consider and compare.

In his thesis “Finding Open options: An Open Source Software Evaluation Model with a Case Study on Course Management Systems" [10], Karin van den Berg proposed an Open Source Software evaluation model. In the model he defined 10 metrics and an assessment process. First he established a selection method, which uses 4 of the 10 metrics and a Linear Weighted Attribute Model to get a quick impression of Open Source Software, filter out undesired ones and provide a short list of candidates. From the short candidate list, he evaluated each metric, assigned a score and weight, and finally used Linear Weighted Attribute Model to calculate the final score.

There are also some other guidelines for choosing Open Source Software, and plenty of improvement work of existing Open Source Software Maturity Model [9] [11][32][33].

II.5 Summary of Existing Maturity Models

Existing Open Source Maturity Models share some common characteristics. Basically they all provide a template which includes a suggested metrics and their weighting factors. An example of such template is shown in Table 2.3.

Table 2.3. Example of maturity assessment template of existing models

Metrics	Maximum score	Real Score	Weighing factor	Weighted score
Number of Lines of Code	10		3	
Support	10		1	
Documentation	10		2	
Integration	10		1	
User Community	10		1	
Market Share	10		1	
Training	10		1	
Total	100		10	

With the maturity assessment template, users need to define their own criteria based on their particular requirements, assess each criterion of the Open Source software, and assign scores. As shown in Figure 2.2, users also need to give a weight factor for each of the criteria, and compute the final maturity score of Open Source project by adding up the weighted scores.

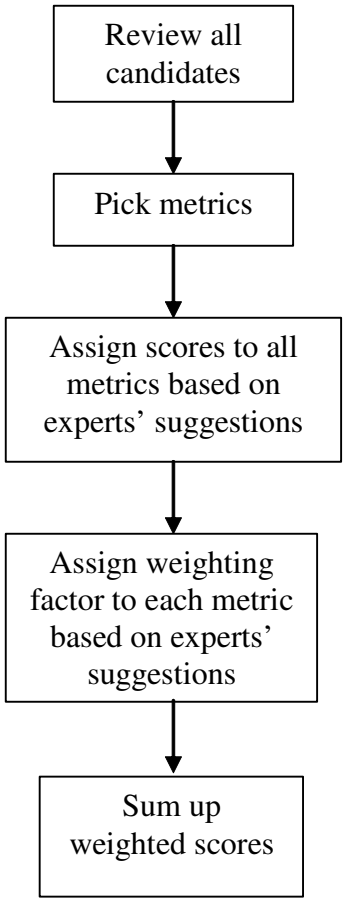


Figure 2.2. User process of computing maturity score based on existing models

Some of the existing Open Source Software Maturity Models provide more information to help users fill in the template and compute the final score. But most of

them require users to assign a reasonable score. Otherwise, users must ask for help from the experienced experts. Table 2.4 shows an example of metrics and threshold for each score level from [12].

Table 2.4. Example of metric ranges and scores from [12]

Metrics	5 – Excellent	4 – Very good	3 – Acceptable	2 – Poor	1 – Unacceptable
Number of Downloads (dow/month)	5000 or more	2000 – 4999	2000 – 1999	500 – 999	0 – 499
Number of Page views (pv/month)	100000 or more	10000 – 99999	1000 – 9999	100 – 999	0 – 99
Number of back links	3000 or more	500 – 2999	100 – 499	50 – 99	0 – 49

CHAPTER III

NEW MATURITY MODEL

All of the Maturity Models summarized in Chapter II are excellent work and very important pioneering work of evaluating the maturity of Open Source Software. They share many similarities. All of them require extensive experience in evaluating software management effectiveness, attributes of development process and their significance, and the product quality. Some of them provide a scoring table that can help users to give a score to each of the metrics.

III.1 Limitations in Existing Models

However, there are four major disadvantages of the existing approaches:

- It leaves too much work to the users. It is very difficult for average users to determine what score they should assign to particular Open Source software for a particular evaluating item. There is no quantitative standard for them to follow. Often they need experts to help them filling in the template, which could be expensive for some users. Furthermore sometimes expert help may not be available to everyone.
- The process of criteria assessment tends to be subjective. Usually a score is assigned based on some subjective preference that is summarized from the

experts' personal experiences. None of the existing models uses statistical data directly. Instead they categorize data based on human experiences. This makes assessment process lack of an objective standard.

- Using categorized data makes the data less informative. For example, Table 3.1 is part of the scoring standard table given in Business Readiness Rating [7]. From Table 3.1, it is clear that if the setup time of the software is 30 minutes to 1 hour, the score should be 3. Consider the case when the setup time is 61 minutes. Although it is only 1 minute more than 1 hour, the score drops from 3 to 2. If this metric happens to be an important one, i.e. it has very high weighting score, then 1 minute makes huge difference. Are 60 minutes setup time and 61 minutes setup time so different? The answer seems to be no. It could even possibly come from the variance from experiment to experiment. The setup time in minutes would represent the nature of setup time better. The same applies to the number of open bugs in the last 6 months for quality indicator and the average volume of messages in the last 6 months from the community indicator group in Table 3.1. The categorized data loses a lot of and sometime important information in the original data.

Table 3.1. Scoring table from Business Readiness Rating [7]

Categories	Metrics	Description	Scoring				
			5 - Excellent	4 – Very good	3 – Accepta-ble	2 - Poor	1- Unacceptable
Usability							
	Time for setup prere-quisites for install-ing open source soft-ware	The time/effort needed to set up a system, with all pre-requisites satisfied. This does not include OS.	< 10 minutes	10 – 30 minutes	30 – 1 hour	1 - 4 hours	> 4 hours
Quality							
	Number of open bugs for the last 6 months	This measures the quality of product usage.	< 50	50 - 100	100 - 500	500 - 1000	> 1000
Community							
	Average volume of general mailing list in the last 6 months	The general mailing list is the place where the community helps itself.	> 720 messages per month	300 - 720 msg per month	150 - 300 msg per month	30 - 150 msg per month	< 30 msg per month

- They are empirical formula, which are determined only by experts' judgments. The weighting factors are obtained based on experiences. There are only positive weighting factors in existing models. This means either no negative attributes are considered or the negative attributes have to be transformed and expressed in a positive manner. These two issues make the existing model either incomplete or less informative. These models are very difficult to use for users who have little experience.

III.2 New Open Source Software Maturity Model

In order to address these problems in the existing approaches, we need to develop a new maturity model. The new model should define criteria which can be readily assessed based on objective standards or statistical data. Instead of assigning a score for each criterion like most existing models do, the author would like to use some quantities in reality to represent the criterion, and provide a set of experimental coefficients that are computed by a statistical procedure using data collected from real Open Source Software repositories. Figure 3.1 shows the procedures of building the new Open Source Software Maturity Model.

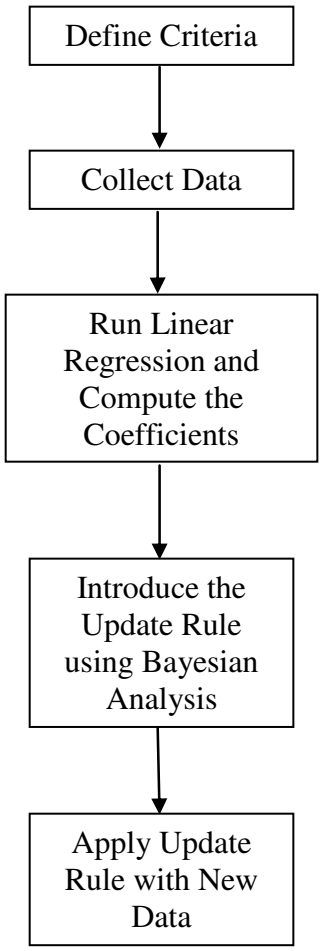


Figure 3.1. The process of building new Open Source Maturity Model

The process of building new Open Source Software Maturity model can be summarized as the following. First, criteria for the new Open Source Software Maturity Model are defined. The importance of each of the criteria and why it is chosen are explained in detail. Second, data is collected from www.freshmeat.net, one of the largest Open Source Software repositories. Then, linear regression model is applied to analyze the data collected in second step. SPSS, one of arguably the best statistical data analysis software, is used to compute the coefficients in the linear regression model.

Once the model is built, the tasks left for users when they want to compare different Open Source projects are just to collect data and feed the data into the model and calculate the final numerical score, as shown in Figure 3.2

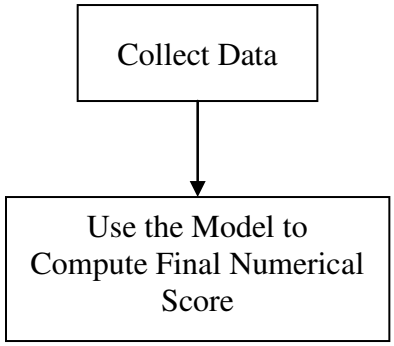


Figure 3.2. Steps to employ the new model to get final numerical score

Comparing Figure 3.2 and Figure 2.1, one can see that the new Open Source Software Maturity Model is much easier to use.

Here we first introduce the evaluation criteria chosen for building the linear regression model and the experiment setup of how to collect data from the existing Open Source Software repository.

III.3 Evaluation Criteria

The existing literature of OSS evaluation provides a good source for the criteria defined here. There are several criteria commonly used by existing models. Based on the literature review, the criteria are defined as following:

Popularity

The more a software product is used, the better chance that the software is good. Unlike proprietary software whose actual usage is measured by counting the number of sold copies, there is no single metric representing accurately how widely it is used for Open Source Software. Open Source Software is downloaded for free; people who download it may not use it. This uncertainty in Open Source Software usage makes it very difficult to measure the exact usage of Open Source Software. Therefore for Open Source Software, it is sufficient to measure the potential usage of the software. Popularity represents the significance of Open Source Software, and in turn represents the potential usage.

Some researchers suggest using the number of downloads [11] to represent the potential use. For example, SourceForge provides the number of downloads, the number of pages viewed, and the number of hits. These statistics can be used to reveal the potential use of the Open Source Software. However, not all repositories provide these

statistics. Using number of downloads as the metric makes it highly depend on the particular features of the Open Source Software repositories [11].

Allessandra Cau et.al. [12] proposed to use the number of backlinks to represent the popularity. In his dissertation, the author chose to measure popularity by PageRank [34]. PageRank computes the relative importance of a webpage by summing up the weighted links from other webpages to it. PageRank has been defined by Google as the following.

“PageRank relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value. In essence, Google interprets a link from page A to page B as a vote, by page A, for page B. But, Google looks at more than the sheer volume of votes, or links a page receives; it also analyzes the page that casts the vote. Votes cast by pages that are themselves "important" weigh more heavily and help to make other pages "important".” [35]

The higher the PageRank of the software project webpage, the larger the number of people who are interested in the software. PageRank is an objective metric to measure the popularity of a software project. The PageRank can be obtained through Google’s Toolbar PageRank feature, as shown in Figure 3.3.

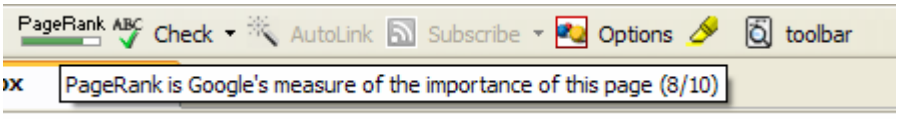


Figure 3.3. Example of PageRank obtained by Google’s Toolbar PageRank checker

Activities

Activities of Open Source Software represents how active the developers and users are in the particular open source software community. Activities are records of what developers have been working on the project and resolutions of users' requests and issue reports. Activities include two types of releases:

- Snapshot release
- Stable release

Usually developers make a release of snapshot in relatively short period of time and a stable release in a relatively long period of time. Snapshot often includes bug fixes and some progress in a new feature development. A few consecutive snapshot leads to a staging release that is called stable release which includes major improvement from the previous stable release and bug fixes.

Both snapshot and stable release include text notes that list all the changes made in the release and usually these notes are documented in a file called "ChangeLog".

When the number of lines of source code becomes bigger and bigger, more activities are needed to fix the bugs in the existing codes. Therefore, the number of release activities per month per thousand lines of source code is used as the activities criterion, i.e., a normalization process. This metric can help people stay away from projects that is abandoned or rarely used.

Quality

High quality codes usually have less number of bugs and issues. Bugs are reported by users in the community. Some software webpage provides a bug tracking tool to help users and developers to keep track of the number of open and closed bugs; others might rely on the software mailing list. Users post the bug report on the software mailing list, and developers or other users can verify and fix it in the software or reply with the workaround methods.

Code quality can be evaluated by the number of bugs reported during a certain period of time. We also need to take into account the total number of messages in the same period of time. To get a more reasonable evaluation of the code quality, we can normalize the number of bug reports against the number of posts during the same period of time.

Support

Support is an indispensable part of a software project. Proprietary software projects usually have the technical support department providing support to users whenever they have any problems with the software. The support may or may not be free of charge. Not every Open Source Software project has devoted people to help users handle their problem. When users have problems with the software project, they often

send the request or the issue report to the mailing list and waiting for replies to help them solve the problem. Support comes from two sources

- community support
- professional support

The community support refers to the support provided by user community. When a user has questions or problems about the software, he or she posts a message on the software mailing list. Other users who encountered and solved the same or similar problems usually reply to the mailing list with his or her solutions. The developers often reply to such messages as well if needed. The quality and response time of this type of support is not guaranteed. Sometimes the users may not get any response at all.

The professional support means the support is provided by professionals. For instance, if a company wants to adopt an Open Source project or want to provide support for it, it may hire professional software developers to provide the support to the users. The professional support is often provided in a timely manner and boasts higher quality. This type of support is also obtained through the project mailing list.

To measure the quality of the support, we use the number of messages per month and normalize it against the number of thousand lines of source code.

Documentation

Documentation explains what features the software provides and how the software should be used. Mature software must be able to deliver high quality

documentation along with the source codes. This way users can find the software easy to use and avoid wasting time to explore the details in the source code.

In a software project, there are usually two types of documentation

- Documentation for end users that explains the features and usage of the software.
- Documentation for the software developers that comes with the source code and explains the parameters and return values of the source code API.

Often software developers are not willing to spend time to write or update documentation for end users, although they keep the development documentation very clear and up-to-date. This may lead to the end users' documentation out of sync with the source code and developers' documentation. Therefore some software projects offer separate resources for each type of documentation and keep them consistent with each other [11].

Documentation comes in two kinds of forms:

- Delivered with the source code and/or posted on the software project website.
- Published books as the manual and examples of usage and features.

It is very often that documentations are published as a book, which means the software project has a fair market share and can attract users to pay for the documentation. It is difficult to transform the value of a published book into the number of lines of documentation. In order to represent the value of both documentation formats, we use two metrics to represent the quality of documentation, the number of lines of the documentation normalized against the number of lines of the source code and the number of published books.

Integration

If one open source project is adopted by a number of other Open Source Software projects, it usually means this open source project complies with the standards [31] and can be easily integrated. This can be considered as the recognition from the software community for the high quality of the software project.

The integration metric is evaluated by the number of projects depending on the project being studied. This number is often provided by the Open Source Software repositories, such as FreshMeat.

III.4 Experiment Setup

For the multiple linear regression model, the most important step is to collect the data from the open source software repositories. The first question to be addressed is the amount of data to be collected to build a meaningful model.

Sample size is a significant factor in the regression algorithm. Apparently better results can be achieved with a larger number of data points. However the number of data points available is always limited. Furthermore the improvement margin may shrink with increasing number of data points. Then the question becomes how to determine the appropriate number of data points. Another related issue is the range of the data points. A set of data points that covers as much as possible of the sample space is more representative and thus a better one.

In determining the appropriate number of data points and data range, there are several issues to be considered.

The first one is the underlying physics or mathematics of the model. If enough information can be provided on the nature of the model, it would be helpful. For instance, if the relation between weight and height of humans is studied, the number of sampling points cannot be 10, 100 or even 1000. The data range of height could range from 0 to 3m. And the data range of weight could range from 0 to 250 kg. However, in the model developed in this study, due to the lack of enough information on the underlying physics or mathematics of the model, the prior knowledge obtained of the nature of the model is limited. This can be understood because the model itself is empirical.

The second one is the indentifiability of the model. The regression model is called identifiable if all the parameters in the model can be determined definitively. In the linear regression model adopted in this investigation, there are 7 parameters corresponding to the 7 metrics and a constant. The number of sampling points must then be 8 at least. However, an identifiable model is not necessarily a convincing model. More sampling points are needed to better represent the sampling space. Also the number of points is a factor that has an impact on the error in the model. With increasing number of points, the impact of the random error associated with the particular software used to build the model is reduced.

The point of the regression algorithm is to make the standard deviation of the model minimum. In the linear model, if the number of sampling points is the same as that of the parameters, the fitting curve is expected to go through all the sampling points

and the standard deviation is zero. However this does not mean the model is perfect. On the contrary, it reflects deficiency. In fact, as the number of sampling points increase, the standard deviation will increase accordingly. However, in a good regression model, the standard deviation will not increase infinitely with the number of sampling points. Rather after some point, the standard deviation tends to converge to a limit. The improvement margin decrease rapidly with the number of sampling points increasing. Therefore numeric experiments can be conducted to find the pattern of standard deviation versus the number of sampling points. If the standard deviation is observed to level out, it means the current number of sampling points might be appropriate.

III.5 Model Validation Method

It is a known problem to validate software evaluation models. It is actually very difficult to strictly prove a model is valid or not. Most models are validated by case studies, such as the famous, wide used COCOMO II [22] software cost estimation model.

The validation plan for this model is using case studies. A number of open source projects will be picked randomly from the open source repository. Data for all the criteria will be collected, and the model will be applied to get the point prediction of maturity score for these projects. The predicted scores will be compared with the user reviews.

CHAPTER IV

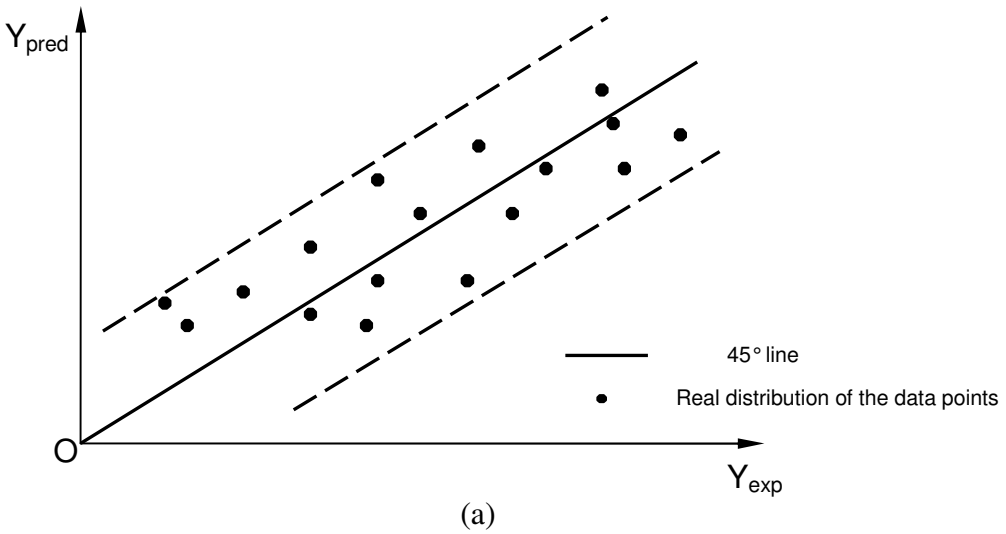
REGRESSION MODEL

Regression is widely used to statistically relate a variable regarded as a random variable, with one or more independent variables, or called predictors. The most commonly used, or the simplest regression is the linear regression, in which the dependent variable is related to the predictors by a linear function. The linear regression model is valid only under certain assumptions. These assumptions ensure that a linear regression model can be built and the dependent variable can be predicted by the model. The coefficients in the linear regression model are obtained by the least square method. In later chapters, it is made clear that these values are but the mean of the coefficients since the coefficients can themselves be regarded as random variables. In this Chapter, the linear regression model for open source software maturity evaluation is built based on 43 baseline software collected from different open source software repositories. After the coefficients are found, the linear model is used to predict the final scores of a number of new software which are not included in the 43 baseline software. The predicted values are then compared with the real user evaluations and they match quite well.

IV.1 Assumptions in the Linear Regression Model

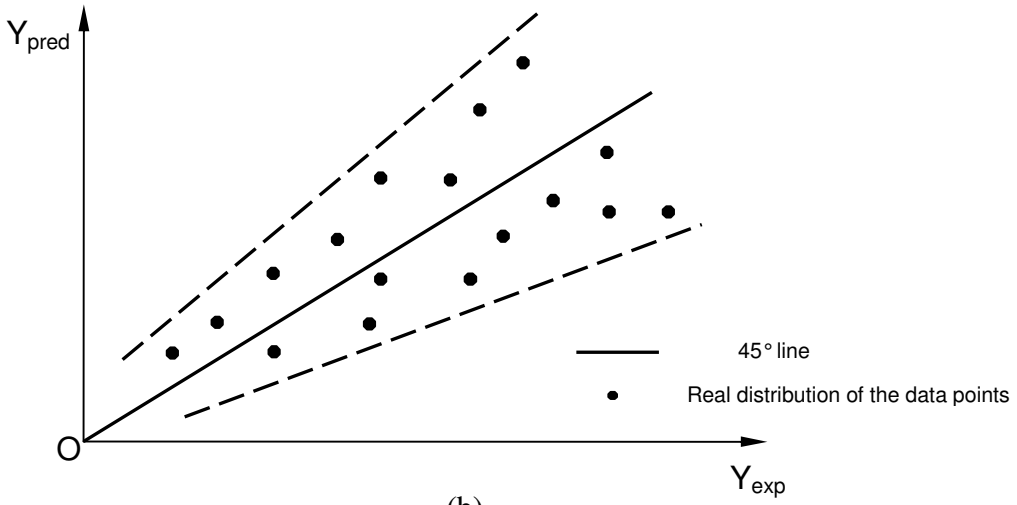
There are several assumptions to be made before a linear regression model can be built. First of all, it is assumed that the experimental data retrieved from experiments can be appropriately related by the assumed mathematical function. Then there are three basic assumptions to be made, which refer to "constant variance", "independence", "normality", respectively [36].

The constant variance assumption means the values of the independent variables do not have an effect on the variance of the dependent variable. The residual plots can be employed to validate this assumption for a particular regression model. The residual is the difference between the predicted value of the dependent variable by the regression model and its experimental value. Figure 4.1 gives three commonly seen scenarios of the residuals in the linear regression. In the case of multiple regressions, it can be alternatively tested by plotting the predicted values against the experimental values.

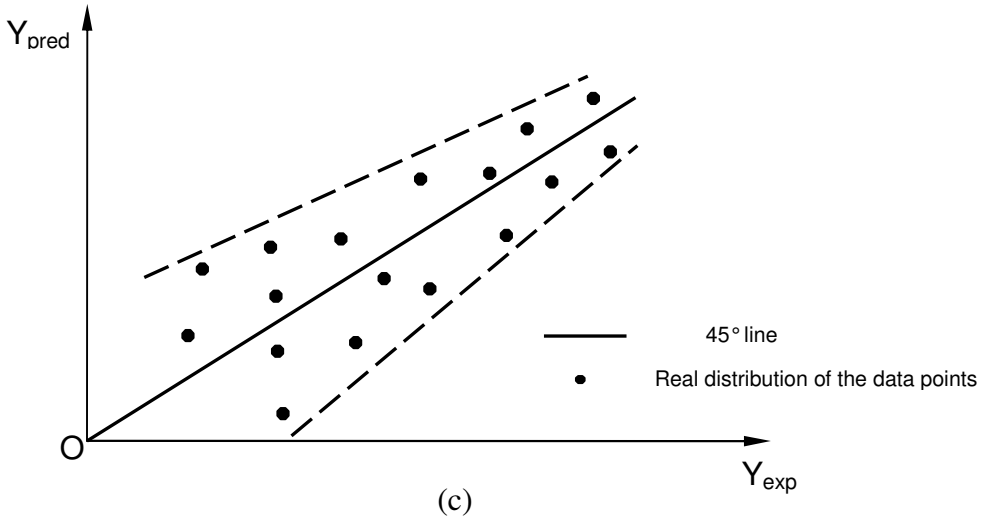


(a)

Figure 4.1. Three scenarios for the residuals of a linear regression model



(b)



(c)

Figure 4.1. Continued

Figure 4.1a shows a pattern in which the residual keeps the same with different values of y, which is a good indication of constant variance. Figure 4.1b and Figure 4.1c, on the other hand, show bad signs of variance change. Only the case in Figure 4.1a can be seen as a good linear regression model.

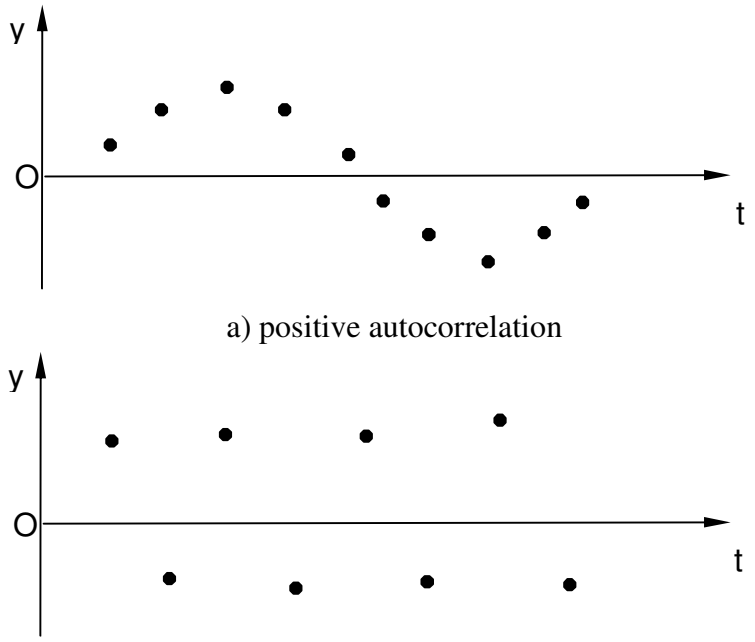
The independence assumption indicates the outcome of dependent variable y is statistically independent. Or alternatively, it can be said that the residuals of the dependent variables are statistically independent. The legitimacy of this assumption can be verified, again, by looking at the residual plots. The residuals must be located randomly. It should not exhibit any periodic pattern. Otherwise it is an indication of autocorrelation and violation of the independency. The independence assumption is most likely to be violated in the time series data, where data can be autocorrelated so that the linear assumption made in the regression model is no longer valid. There are two types of autocorrelation, namely, positive autocorrelation and negative autocorrelation. In the positive autocorrelation case, the data shows a periodic pattern. In the negative autocorrelation case, the data shows an alternating pattern over time. Figure 4.2 shows the two cases of autocorrelation. In statistics, the Durbin-Watson Test is used to check if the data has autocorrelation pattern. The Durbin-Watson Test is defined by the following index [37].

$$d = \frac{\sum_{t=2}^{n} (e_t - e_{t-1})^2}{\sum_{t=1}^{n} e_t^{\,2}} \tag{4.1}$$

where, e_t is time ordered residual at time t, e_{t-1} is time ordered residual at time $t-1$, n is the total number of data points.

Autocorrelation happens when the process has a certain built-in mechanism. For data collected from different users to evaluate the open-source software, it is not likely that the autocorrelation would happen. In fact, since the sequence of the data points, each

of which represents an open source software, is not important in the analysis, the *d* value obtained from Equation (4.1) would vary with how the data points are ordered. The order of the data points, however, is quite a random event. This indicates that autocorrelation is certainly not the case in this investigation.



a) positive autocorrelation

b) negative autocorrelation

Figure 4.2. The two cases of autocorrelation

The normality assumption states any of the dependent variables has a normal distribution. Or alternatively, for any value of the independent variable, the error of the regression model has a normal distribution. To verify this assumption, the histogram of the residuals must look reasonably like the bell shape of the normal distribution with a mean of zero.

The normality assumption, however, can be verified through a quantitative procedure, namely the normal plot. In the normal plot, the residuals must be sorted in the ascending order. Next the number corresponding to each of the residual $(3i-1)/(3n+1)$ is calculated, in which i is the index of the residuals and n is the total number of data points [38]. Each of these numbers corresponds to a point on the standard normal distribution graph such that the area below this point is equal to the number. From here, another sequence of points is obtained. Then the sorted residuals are plotted against the obtained points. If the plot appears to be a straight line, the normality of the dependent variables is confirmed.

From the above description, it can be understood that all the assumptions cannot be verified on a prior basis because there is no error or residual information available until the model is established. All these assumptions can only be verified after the parameters in the linear regression model are all found and the predicted value of the dependent variable can be calculated from the model.

VI.2 The Linear Regression Model

The linear regression model is described by

$$y = \sum_{i=1}^{M} \beta_i x_i + c \tag{4.2}$$

In Equation (4.2), β_i represents the coefficient to be determined for the ith factor and x_i is the score for that factor. M is the total number of factors, which is 7 in this case. y

represents the final score for a particular software. Assume the number of software investigated is N. Equation (4.2) can be written in the matrix form for the pool of software. Assume that Y is the $N \times 1$ column vector that contains the total scores for the software evaluated. X is the $N \times M$ matrix whose rows contain the score for each of the factors for particular software. **B** is $M \times 1$ column vector that contains the coefficients. **I** is a column vector with all entries equal to 1. Then Equation (4.2) can be rewritten as

$$\mathbf{Y} = \mathbf{XB} + c\mathbf{I} \tag{4.3}$$

At this point, it is appropriate to combine the constant term and the coefficient term. To do this, add an additional column with all its entries to be 1 to the matrix **X** and add c to the B vector. This means matrix **X** which represents the collected data will always have "1" corresponding to the "coefficient" *c*. The new equation takes the form

$$\mathbf{Y} = \mathbf{XB} \tag{4.4}$$

The error vector can be expressed as

$$\mathbf{e} = \mathbf{Y} - \mathbf{XB} \tag{4.5}$$

The norm of the error vector is

$$e = (\mathbf{Y} - \mathbf{XB})^T (\mathbf{Y} - \mathbf{XB}) \tag{4.6}$$

To make this norm the minimum, take the derivative of it with respect to vector B and make it zero. From here, the vector B can be obtained.

$$\mathbf{B} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{4.7}$$

Before the model can be established, there is an important issue to be considered. That is the identification of outliers. An experimental data point that significantly deviates from the rest of the points is called an outlier. If the least square estimate gives

substantially different results after a data point is removed, this point is said to be influential. An outlier is not necessarily influential. As shown in Figure 4.3, Data Point #1 is an outlier with respect to its *x* value, but it is not outlying with respect to its *y* value. Furthermore, there are several other data points that have similar *y* values and their x values are in the range. Therefore, Data Points #1 may not be influential. The same situation happens to Data Point #2. Data Point #3, however, can be an influential outlier since both its x and y values are out of range and it is not consistent with the trend of the majority of the data points.

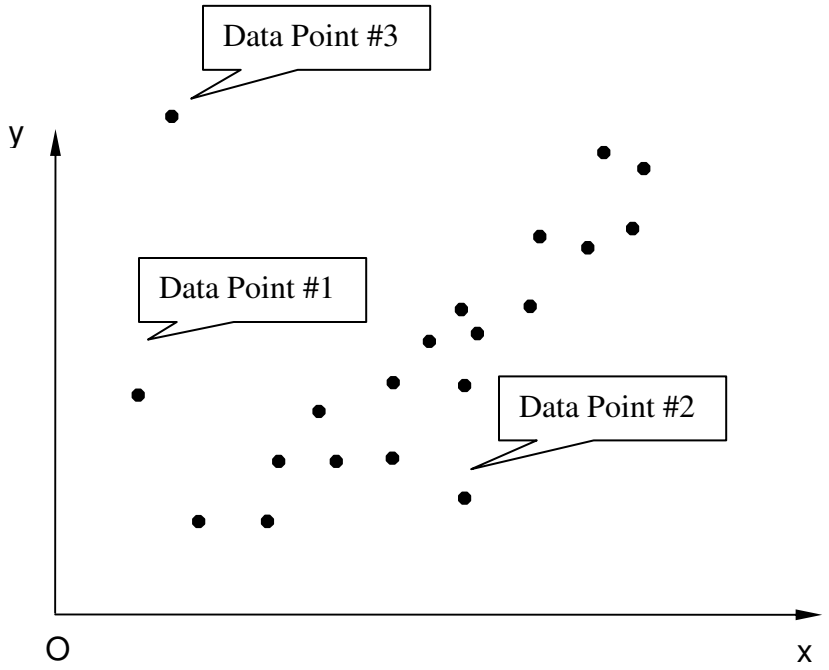


Figure 4.3. Identify an outlier

Besides visual inspection, there are more scientific ways to identify an outlier and determine if they are influential. To determine if a point is an outlier with respect to

its x value, the Leverage Value is used [36]. The Leverage Values are the diagonal entries of the Hat Matrix which is defined by

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{4.8}$$

where matrix **X** is the same matrix as defined in Equation (4.3), the superscript *T* stands for transpose of the matrix and superscript exponential index -1 stands for inverse of the matrix. The leverage value of the ith observation can be shown to be equal to

$$h_{ii} = X_i(\mathbf{X}^T\mathbf{X})^{-1}X_i^T \tag{4.9}$$

where X_i is the vector containing the values of the independent variables in the *i*th observation. It can be shown that the leverage value is always between 0 and 1. A leverage value is considered to be large if it is substantially greater than most of the other leverage values. At this point, it has to be pointed out that in this study the observations of the last two independent variables, Number of Books and Number of Projects, are major causes of outliers. For most of the software, the observation of these two independent variables is zero, and it is nonzero for just a few software. By intuition, this leads to the outlying situation. However, since it is necessary to include the influence of these two factors, they are excluded from the outlier evaluation.

To determine if an observation is an outlier with respect to its y value, the so-called studentized residual, among other indexes, is used. The studentized residual [39] is defined by

$$\eta = \frac{e_i}{s\sqrt{1-h_{ii}}} \tag{4.10}$$

where e_i is residual of the *i*th observation, s is the standardized error of the regression model. If the value of studentized residual of one observation is significantly larger than that of the other observations, it is regarded as an outlier.

If it is concluded that an observation is an outlier, it is left to determine its influence. Cook's Distance Measure (CDM) is used to quantify the influence. CDM for the ith observation is defined as

$$CDM_i = \frac{(\mathbf{B} - \mathbf{B}_i)^T \mathbf{X}^T \mathbf{X} (\mathbf{B} - \mathbf{B}_i)}{ks^2} \tag{4.11}$$

where $\mathbf{B}$ is coefficient vector obtained including the *i*th observation and $\mathbf{B}_i$ is the coefficient vector obtained excluding the *i*th observation. *k* is the number of coefficients, which is 8 in this study. If CDM_i is large, then the *i*th observation can be considered as influential. The F-distribution is used to indicate if the value of CDM is large. The rule of thumb is as the following [40]. If CDM_i is larger than $F_{[.80]}^{(k,n-k)}$ (20% of the F-distribution that having k and n-k degrees of freedom), then the *i*th observation is not considered influential. If CDM_i is smaller than $F_{[.50]}^{(k,n-k)}$, then the *i*th observation is considered influential. When CDM_i falls in between, the influence is considered larger when it comes closer to $F_{[.50]}^{(k,n-k)}$.

From Equation (4.10) and Equation (4.11), it can be seen that the influence of an outlier can be only determined after the model is built. Therefore, to determine if an observation is an influential outlier, a trial and error procedure has to be followed. This is a very tedious procedure. The 43 observations or open source software on which the baseline model is based are chosen through this procedure.

Another issue that needs to be addressed is the collinearity between the parameters. In the linear regression model, the independent variables are supposed to be statistically independent [36]. To check this, the correlation coefficients between different independent variables must be zero in the mathematical sense or reasonably small in the practical sense. Although it is not true that when the independent variables are not linearly correlated, they must be statistically independent, the linear correlation coefficients at least can be used to check in one way if the statically independence assumption is violated. The Table 4.1 shows the correlation coefficients between the independent variables. From this table, it is seen that the linear correlation between the independent variables is weak.

Table 4.1. The correlation coefficients between the independent variables

	v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*
v1	1	-0.0448	-0.2548	0.0713	0.1295	0.1684	0.3395
v2	-0.0448	1	-0.05	0.2783	-0.2066	0.0799	-0.2039
v3	-0.2548	-0.05	1	-0.1279	-0.03	-0.053	-0.1059
v4	0.0713	0.2783	-0.1279	1	-0.1246	-0.0167	0.0057
v5	0.1295	-0.2066	-0.03	-0.1246	1	-0.1987	-0.0103
v6	0.1684	0.0799	-0.053	-0.0167	-0.1987	1	-0.0357
v7	0.3395	-0.2039	-0.1059	0.0057	-0.0103	-0.0357	1

The data collected from the 43 open source software are listed in Appendix A. From the data, the X matrix and Y vector can be formed and the regression coefficients

are found. Table 4.2. shows the values of the coefficients and the corresponding standard errors and the 95% confidence intervals.

Table 4.2. The results of linear regression

	Coefficient	*Standard Error*	*Confidence Interval 95%*	
			Lower	*Upper*
c	8.16654586	0.216551788	7.726922357	8.606169362
Page Rank	0.034405181	0.035832458	-0.038338575	0.107148938
activities/month/TLOC	0.707593224	0.64868034	-0.609297878	2.024484326
Number of Bugs / Number of Messages	-0.282736641	0.167454568	-0.622687487	0.057214204
Number of Messages/TLOC	0.037728903	0.08783903	-0.140593808	0.216051615
Documents/TLOC	0.003657083	0.002843056	-0.002114626	0.009428793
Number of BOOKs	0.278552285	0.267614883	-0.264734812	0.821839381
Number of projects depending on this one	0.00890032	0.004769016	-0.000781298	0.018581937

It shall be understood that the relative magnitudes of the coefficients cannot be used to characterize the strength of the factors in evaluating the software. For example, the coefficient of the factor Documents is two orders of magnitude less than that of the factor Activities. This does not mean Documents has less influence on the final score than Activities. It is that the number of Documents happens to be large compared with the number of Activities normalized against time and software size. The reason for not

normalizing values of the factors is that a normalization procedure is inevitably related only to the current set of collected data. The current normalization has nothing to do with any future incoming data. Therefore, with any incoming data for future applications, the normalization has to be modified to include the future data. This causes volatility in the model and thus should be avoided. It is also seen that the coefficients are positive for all factors except Number of Bugs. This negativity is reasonable. The quality of the software shall be regarded higher with fewer bugs reported. The negative coefficient is exactly a reflection of this fact.

IV.3 Verification of the Three Assumptions

In this section, the three assumptions in Section IV.1 are examined now that the model is established and predicted values and errors can be obtained. In the following table, the residuals from the regression model, which is the difference between the predicted value of the dependent variables and the corresponding user review score, is listed. It can be seen that the maximum percentage of error is within 5%.

Table 4.3. Residual of the regression model

Software	**User reviews**	**Predicted Value**	**Residual**	**Percentage(%)**
1	9.1	8.968030217	0.13197	1.4502174
2	8.61	8.409854873	0.200145	2.32456594

Table 4.3. Continued

Software	User reviews	Predicted Value	Residual	Percentage(%)
3	8.73	8.73	5.33E-14	6.1043E-13
4	8.92	8.511371307	0.408629	4.58103916
5	8.77	8.549252985	0.220747	2.51706972
6	8.4	8.172289811	0.22771	2.71083558
7	8.38	8.366706401	0.013294	0.15863484
8	8.15	8.13752295	0.012477	0.15309264
9	7.8	8.316494371	-0.51649	-6.6217227
10	8.12	8.516585021	-0.39659	-4.884052
11	8.54	8.622870998	-0.08287	-0.9703864
12	8.53	8.55758725	-0.02759	-0.3234144
13	8.52	8.574964502	-0.05496	-0.6451233
14	8.5	8.31012779	0.189872	2.23379071
15	8.47	8.4122926	0.057707	0.68131523
16	8.43	8.367180124	0.06282	0.74519426
17	8.39	8.441229082	-0.05123	-0.6105969
18	8.34	8.394689691	-0.05469	-0.6557517
19	8.26	8.473386431	-0.21339	-2.5833708
20	8.19	8.425229823	-0.23523	-2.872159
21	8.83	8.57054552	0.259454	2.93832933
22	8.79	8.366991779	0.423008	4.81238021
23	8.75	8.49351553	0.256484	2.93125109
24	8.68	8.50279298	0.177207	2.04155552

Table 4.3. Continued

Software	User reviews	Predicted Value	Residual	Percentage(%)
25	8.68	8.390230272	0.28977	3.33836093
26	8.66	8.627377424	0.032623	0.37670411
27	8.5	8.43210743	0.067893	0.79873612
28	8.49	8.706087045	-0.21609	-2.5451949
29	8.46	8.628414988	-0.16841	-1.9907209
30	8.45	8.540939487	-0.09094	-1.0762069
31	8.4	8.316379426	0.083621	0.99548302
32	8.39	8.459808167	-0.06981	-0.8320401
33	8.33	8.471718195	-0.14172	-1.7012989
34	8.28	8.408617026	-0.12862	-1.5533457
35	8.19	8.455843011	-0.26584	-3.2459464
36	8.19	8.444858249	-0.25486	-3.1118223
37	8.1	8.436003885	-0.336	-4.1481961
38	8.09	8.410178514	-0.32018	-3.9577072
39	7.99	8.210626622	-0.22063	-2.7612844
40	7.97	8.172043786	-0.20204	-2.5350538
41	8.78	8.478783117	0.301217	3.43071621
42	8.69	8.311327038	0.378673	4.35757149
43	8.66	8.407144281	0.252856	2.919812

First the constant variance assumption is checked. On Figure 4.4, the predicted values are plotted against the experimental ones, i.e., the score given by the user reviews.

It is seen the data points are distributed along the 45° line. Also the data points lie in a parallel band. There is no fanning out or funneling in of the band.

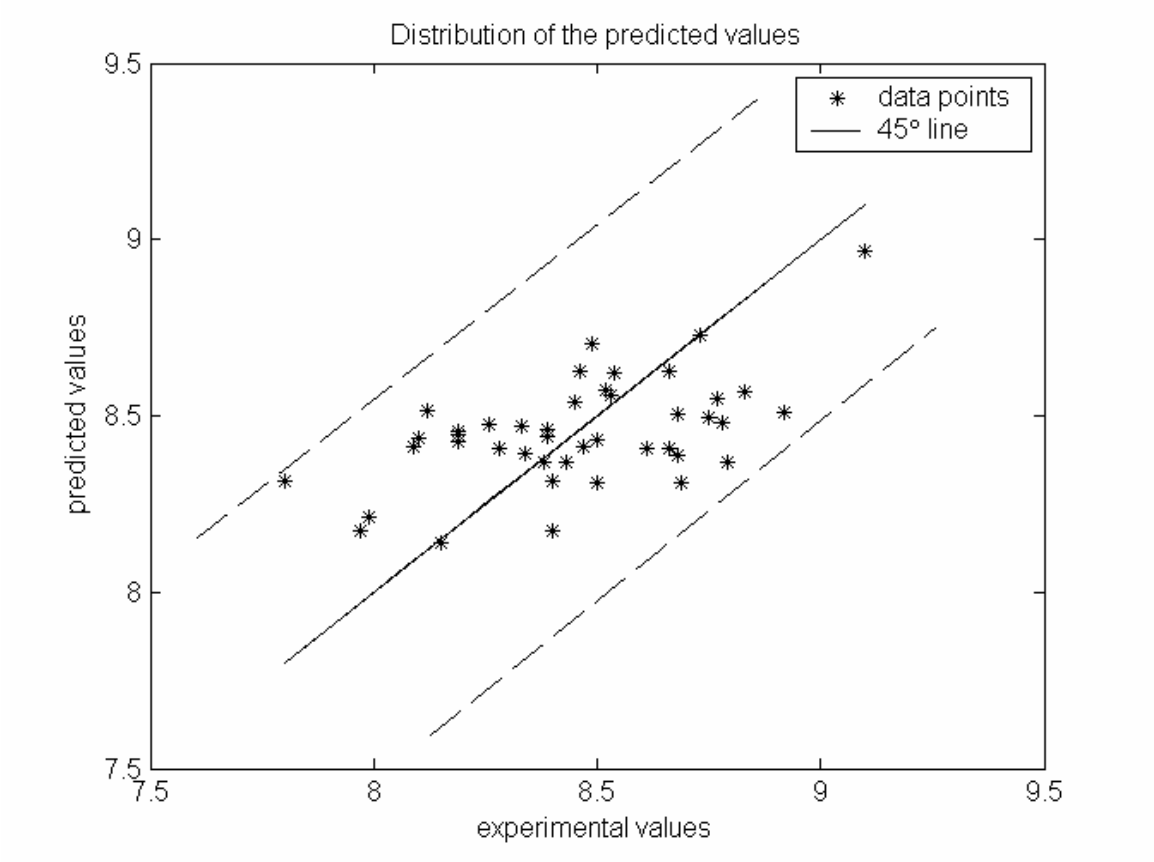


Figure 4.4. The distribution of the predicted values against the experimental values

As stated before, there is no need to verify Assumption 2 since no time sequence is involved in this study. Next, Assumption 3 is verified through the normal plot. In Figure 4.5, it can be seen that the plot does not deviate from a straight line very much. The maximum deviation appears to happen at the end of the line. But the overall linearity of the plot is obvious. The linearity correlation coefficient of the curve is

0.9941, which is a good indication of the linearity [41]. This verifies the normality assumption.

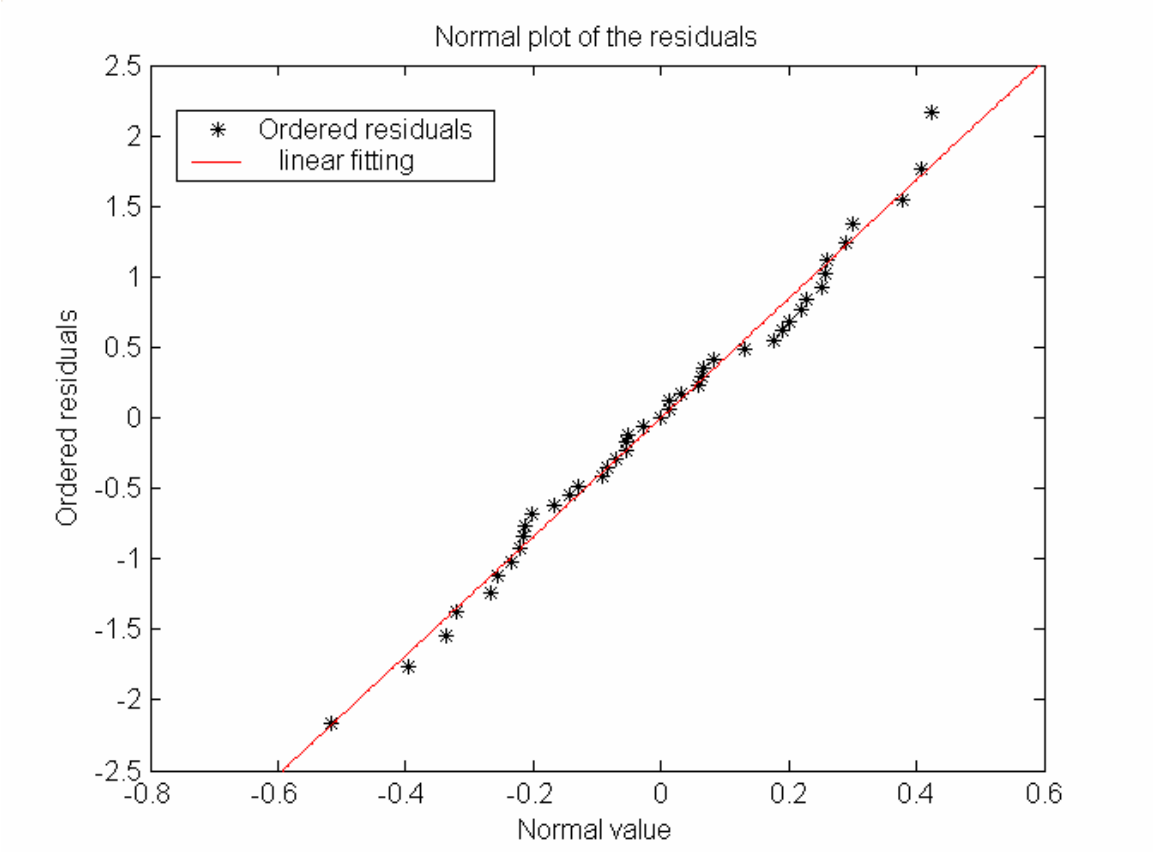


Figure 4.5. Normal plot of the ordered residuals

IV.4 Model Testing

It is realized that the distribution of the coefficients, as indicated by its standard error and 95% confidence interval as shown in Table 4.3. leaves room for improvements. However, since the objective of this study is to establish an objective and quantitative method to evaluate open source software, it is more important to examine the model

against new software cases. To do this, the final scores of a list of open source software, which are not included when the model is calculated, are obtained through this model and the results are compared with those from the user reviews.

Table 4.4. The predicted values from the model and the user review values

Project Name	**User Review**	**Prediction**	**Error (%)**
KCachegrind	8.58	8.658	0.913
Cscope	8.54	8.877	3.951
OpenMotif Everywhere	8.48	8.409	-0.835
UPS	8.45	8.520	0.829
Interverse	8.5	8.310	-2.234
dircproxy	8.63	8.813	2.116

From Table 4.4, the error of the predicted value from the model is within 5% compared with user reviews. This is fairly good prediction. It has to be pointed out that the user review scores do not necessarily reflect an objective estimate. Therefore the results from the comparison between the predicted value and the user score may vary from software to software. To maximize the objectivity, the regression model, or more precisely, the coefficients in the model, need to be updated continuously according to new available software or data, which is the topic of Chapter V.

CHAPTER V

MODEL UPDATING BY BAYESIAN STATISTICAL ANALYSIS

In this chapter, the method to update the regression model according to new incoming data is described. As shown in the previous chapter, the coefficients, or more appropriately called parameters, in the linear regression model are obtained through least square technique. Now it should be explicitly stated that they are only the mean of these parameters when they are considered to be random variables. In statistics, this is called a point estimate of the parameters. The point estimate always depends on the currently available data. To update the model parameters, new data has to be added to the data set that is used to obtain the current parameters. This makes the implementation difficult because the user who owns the new data does not necessarily have the knowledge of the data set used to obtain the current model. All that the user owns is the statistical parameters in the current model and the new data. Therefore it is necessary to build an updating procedure that depends only on the current model parameters and the new incoming data. This can be achieved through Bayesian statistics. In the following sections, the concept of Bayesian analysis is introduced and how its application to the model developed in the previous chapter is presented. It begins with the Bayesian theorem. Based on the Bayesian theorem, the prior and posterior distribution of the coefficients in the linear regression model for single and multi-variable cases are

discussed and then an updating rule is established for further updating process on the linear regression model.

V.1 Bayesian Theorem

In statistics, there are basically three different approaches in probability estimate, namely frequentist, Bayesian and likelihood. In the frequentist approach, it is assumed that the experiment on the outcomes of a certain random variable can be performed as many times as needed so that the probability can be estimated based on the number counts. And it is assumed when the number of times of the experiments being carried out approaches infinity, the probability calculated from the number counts converges to the "true" likelihood. In the Bayesian approach, an assumed prior distribution of the random variable is combined with observed values to generate a posterior distribution [42]. So the essential elements in Bayesian approach are prior distribution, observed data and posterior distribution. The bridge to connect the prior and posterior distribution is the observed data. Alternatively, it can be said that the prior distribution is "updated" through the observed data. The likelihood approach takes out the prior element from the Bayesian approach so that the parameters are estimated directly from the observed data according to the statistical model. Apparently the likelihood approach becomes difficult if the user has no knowledge of previous data set.

Next, the Bayesian inference is introduced following Box and Tiao's [43] description. Assume a random variable $\mathbf{y}$ with a probability density function of $p(\mathbf{y})$. The

parameters in the model are $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots\ldots \theta_k)$. The distribution of $\boldsymbol{\theta}$ is $p(\boldsymbol{\theta})$. Then from Bayesian rule, the conditional probability density function $p(\mathbf{y}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta}|\mathbf{y})$ are related by Equation (5.1)

$$p(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) = p(\mathbf{y}, \boldsymbol{\theta}) = p(\boldsymbol{\theta} \mid \mathbf{y}) p(\mathbf{y}) \tag{5.1}$$

The conditional probability density function is then

$$p(\boldsymbol{\theta} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathbf{y})} \tag{5.2}$$

In Equation (5.2), the probability density function $p(\mathbf{y})$ can be obtained by integrating the conditional probability density function $p(\mathbf{y}|\boldsymbol{\theta})$, following the total probability rule. If $\boldsymbol{\theta}$ is a set of continuous random variables, the probability density function $p(\mathbf{y})$ can be obtained by

$$p(\mathbf{y}) = \int p(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{5.3}$$

When the factor $\frac{1}{p(\mathbf{y})}$ in Equation (5.2) is replaced by a constant c, the Bayesian theorem is derived.

$$p(\boldsymbol{\theta} \mid \mathbf{y}) = c p(\mathbf{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \tag{5.4}$$

An examination of Equation (5.4) provides some interesting observations. It relates the probability density function $p(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta}|\mathbf{y})$. When $p(\boldsymbol{\theta})$ is known, $p(\boldsymbol{\theta}|\mathbf{y})$ can be calculated through Equation (5.4). This means one's level of knowledge about the distribution of the parameters $\boldsymbol{\theta}$ has increased given new information about $\mathbf{y}$. In Bayesian statistics, $p(\boldsymbol{\theta})$ is regarded as a *prior* distribution of the parameters $\boldsymbol{\theta}$ and $p(\boldsymbol{\theta}|\mathbf{y})$ is regarded as a *posterior* distribution of the parameters $\boldsymbol{\theta}$. In this light, Equation (5.4)

provides a method to update the distribution of the parameters. Furthermore, the conditional probability density function $p(\mathbf{y}|\boldsymbol{\theta})$ can be viewed as a function of $\boldsymbol{\theta}$, given new data of $\mathbf{y}$. This function is called likelihood function by Fisher [44]. After this, the Bayesian theorem can be written as

$$p(\boldsymbol{\theta} \mid \mathbf{y}) = cL(\boldsymbol{\theta} \mid \mathbf{y})\, p(\boldsymbol{\theta}) \tag{5.5}$$

Now the coefficient c can be regarded as a normalization factor to ensure that the integration of $p(\boldsymbol{\theta}|\mathbf{y})$ over the entire domain is one. This establishes the following relation.

$$p(\boldsymbol{\theta} \mid \mathbf{y}) \propto L(\boldsymbol{\theta} \mid \mathbf{y})\, p(\boldsymbol{\theta}) \tag{5.6}$$

Apparently, this relation can be repeated a number of times and leads to

$$p(\boldsymbol{\theta} \mid \mathbf{y}_N, ...\mathbf{y}_2, \mathbf{y}_1) \propto L(\boldsymbol{\theta} \mid \mathbf{y}_1) L(\boldsymbol{\theta} \mid \mathbf{y}_2) ... L(\boldsymbol{\theta} \mid \mathbf{y}_N)\, p(\boldsymbol{\theta}) \tag{5.7}$$

V.2 Updating Normal Distributions by Bayesian Theorem

In this section, the Bayesian theorem is applied to a one-dimensional Normal distribution example raised by Box and Tiao [43]. The purposes are twofold. First this demonstrates the updating procedure. Secondly the parameters in the linear regression model developed in this study have multivariate Normal distribution. The one-dimensional distribution example would help to understand the multivariate ones.

Suppose there are two physicists A and B who are trying to get a more accurate estimate of a certain dimensionless physical constant θ. Lack of prior knowledge of what distribution of θ, the physicists just assume it has a Normal distribution, i.e.,

$\theta \sim N(\bar{\theta}, \sigma^2)$. $\bar{\theta}$ is the mean and σ is the standard deviation. This means the probability density function would be the Normal function as listed in (5.8)

$$f(\theta) = (2\pi\sigma^2)^{-1/2} \exp\left[-\frac{1}{2}(\frac{\theta - \bar{\theta}}{\sigma})^2 \right] \tag{5.8}$$

Physicist A, who is more experienced, decides that the mean should be 900 and standard deviation should be 20 while Physicist B, who is less experienced, decides that these two numbers should be 800 and 80. Apparently B is less certain about the exact value of the constant. And his estimate of the constant is smaller than that of Physicist A.

Assume that through an objective experimental method, an observation is made of a value *y* that has a Normal distribution with mean θ and standard deviation σ_0 and $\sigma_0 = 40$. The impact of this observation on the physicists estimates of the mean θ can be evaluated. The likelihood of the observation of value *y*, since it has a Normal distribution, is

$$L(\theta \mid y) = \exp\left[-\frac{1}{2\sigma_0^2}(\theta - y)^2 \right] \tag{5.9}$$

According to Equation (5.5), the conditional probability of θ is found as the following.

$$p(\theta \mid \mathbf{y}) = cL(\theta \mid \mathbf{y})p(\theta) = c\exp\left[-\frac{1}{2\sigma_0^2}(\theta - y)^2 - \frac{1}{2\sigma^2}(\theta - \bar{\theta})^2 \right] \tag{5.10}$$

From Equation (5.10), the mean and standard deviation of the distribution $p(\boldsymbol{\theta}|\mathbf{y})$ are

$$\bar{\theta}_{new} = \frac{1}{w_0 + w_1}(w_0 \bar{\theta} + w_1 y) \tag{5.11a}$$

$$\frac{1}{\sigma_{new}} = w_0 + w_1 \tag{5.11b}$$

where $w_0 = \frac{1}{\sigma_0^2}$ and $w_1 = \frac{1}{\sigma^2}$.

Now the impact of the observation y on the distribution of θ can be quantified for both of the physicists. Figure 5.1a through Figure 5.1d show the prior distribution $p(\boldsymbol{\theta})$ and the posterior distribution $p(\boldsymbol{\theta}|\mathbf{y})$ with different observation values of y for both physicists. Several observations can be made from these figures. First, the difference between the two physicists' posterior estimates is smaller than that between their prior estimates. This can be concluded by comparing the means and standard deviations of the posterior distributions between Physicist A and B. Secondly, Physicist B seems to have learned more from the experiment than Physicist A. This can be seen by comparing the difference in the means and standard deviation before and after the experiment. Thirdly, Physicist B is more certain about the distribution of θ after the introducing the observed value of y. This can be seen from the fact that the standard deviation of the posterior distribution for Physicist B is smaller now. From these observations, a couple of conclusions can be made. First, experiments help to downplay the influence of personal experiences on the estimate. Secondly, it suggests that personal experiences are most likely result from previous experiments, not unfounded guess. In a word, experiments or data will help to maximize objectivity in one's estimate. This is an important conclusion and it is exactly why an updating rule needs to be established to continuously improve the linear regression model.

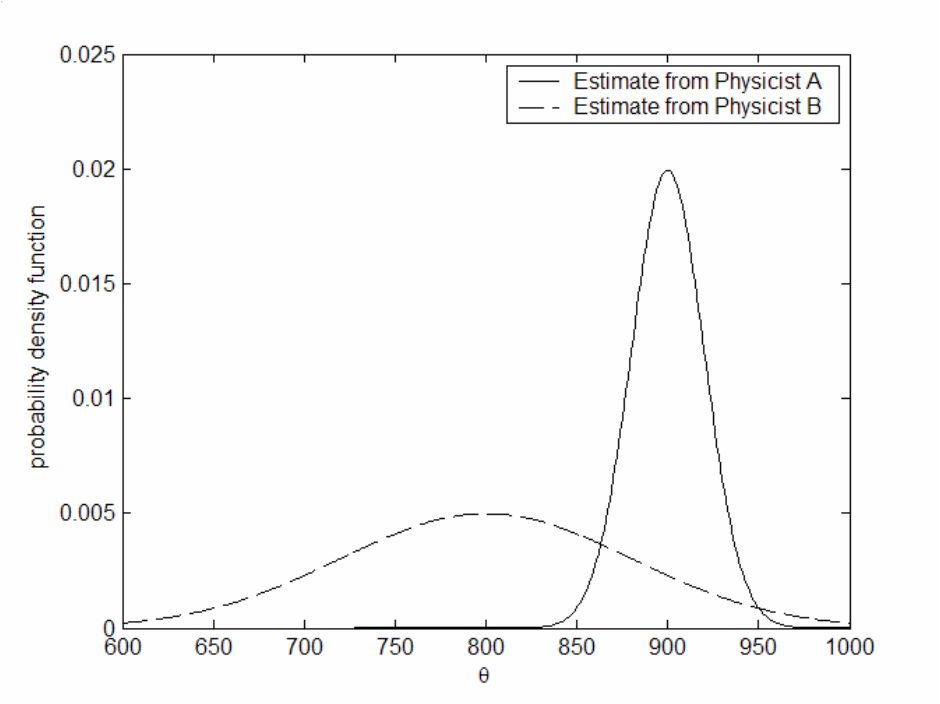


Figure 5.1a. The prior distribution of θ according to the two physicists

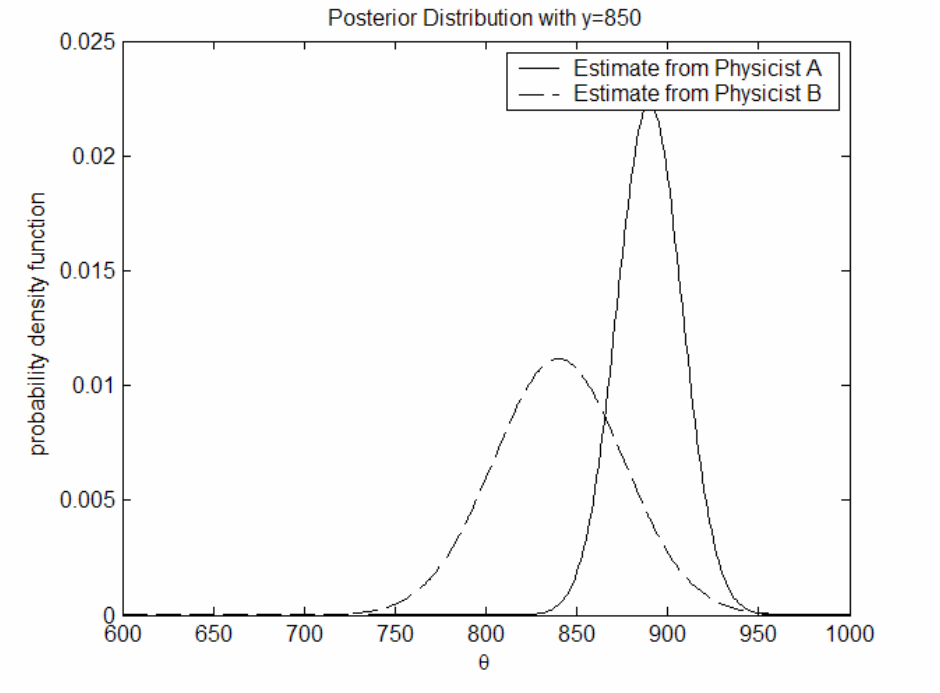


Figure 5.1b. The posterior distribution of θ according to the two physicists with y = 850

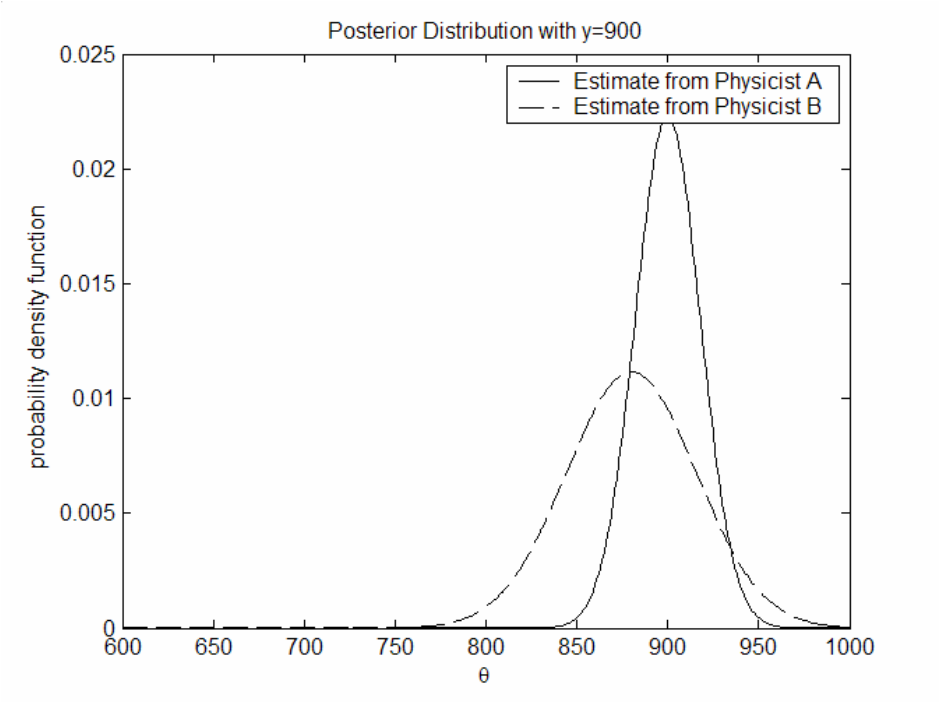


Figure 5.1c. The posterior distribution of θ according to the two physicists with y=900

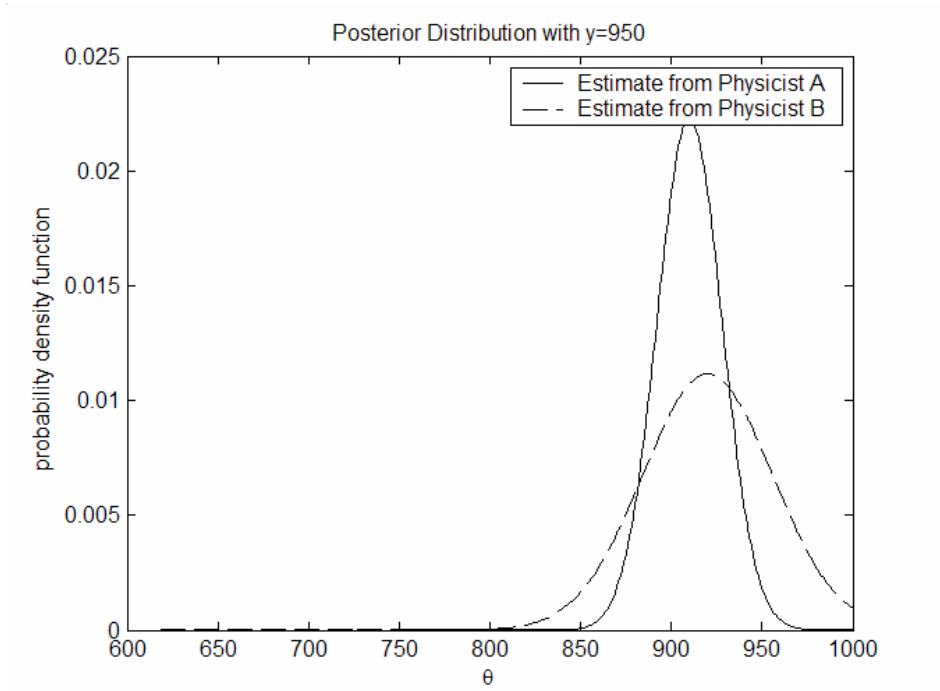


Figure 5.1d. The posterior distribution of θ according to the two physicists with y=950

Now it is clear that the observed value or new data would help to significantly change one's knowledge about the value of a random variable. This is especially true when one's prior knowledge about the random variable is limited. This reminds one to think about the effects of a series of new data on one's estimate. Take the same example and the estimate of Physicist B is updated by a series of new incoming data of 800, 850 and 900. Figure 5.2 shows the prior and updated distributions. It is observed that the uncertainty in the distribution of θ, as indicated by its standard deviation, is reduced with each updating action when the observed values move consistently in one direction. This also reminds one that with one or two observations, the impact of prior distribution can be large. But when more and more observations become available and if these observations are consistent, the impact of prior distribution becomes less so the posterior distribution would correctly represent or converge to the "true" distribution reflected by all these observations.

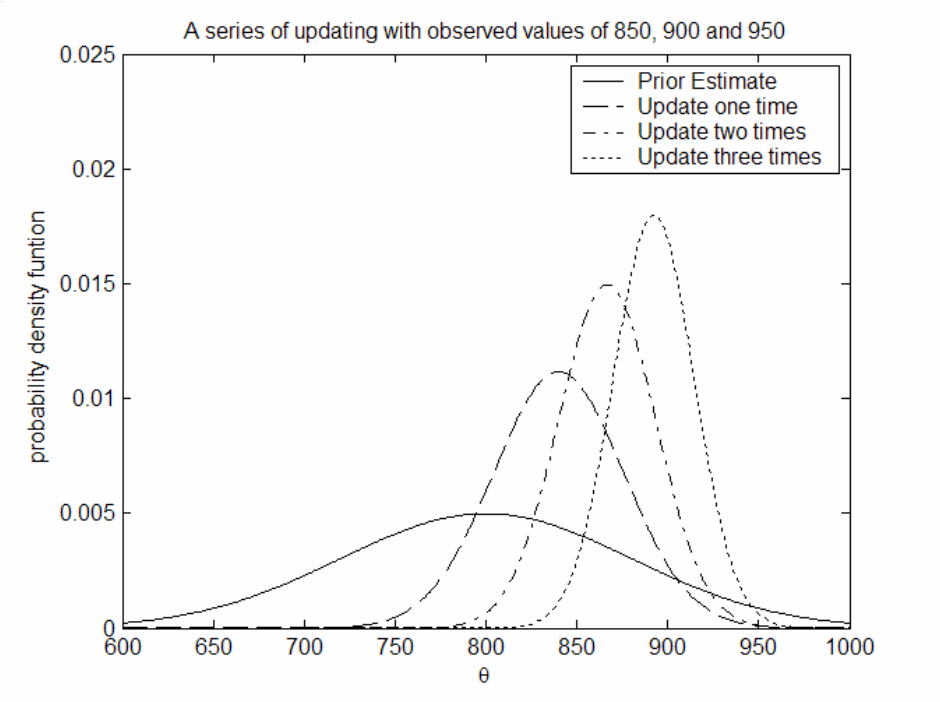


Figure 5.2. Series of updating on Physicist B's estimate

It is worth noting that in the previous discussion, the standard deviation of the likelihood function is smaller than that of the prior distribution. This is called likelihood dominancy [45]. The convergence of the updating distribution to the true distribution depends on likelihood dominancy. In scientific inference, this assumption is a common situation. Otherwise, it is meaningless to update the prior distribution with the "less certain" data.

V.3. Likelihood Dominancy

Likelihood dominancy is an important concept in the science and engineering fields, which are mostly empirical and experiment-dependent. Due to its significance in this study, some discussions of this concept are necessary. Generally speaking, likelihood dominancy means a scientist's prior knowledge of a parameter is less important than the data from experiments. A scientific experiment would not be performed unless the data collected from the experiment would be considered more accurate. From the discussion in the previous section, it can be observed that likelihood is associated with data or experiments. The choice of a prior distribution, however, is more subjective and may vary from person to person. Therefore likelihood dominancy is actually a requirement of objective modeling. It ensures personal opinions, as reflected by the prior, will be gradually eliminated and the model is improved through the introduction of experimental data. In the case study of open source software evaluation, since there is no underlying mathematics to justify any prior model and all models are basically a guess, the successful model is not necessarily a good guess in the first place. More important, the model is able to improve itself with new incoming data. This is exactly why the parameters in the linear regression model developed in Chapter IV needs to be updated by the Bayesian analysis.

Figure 5.3 shows graphically the dominancy of likelihood. Box and Tiao [43] had given a general description, it says "a prior which is dominated by the likelihood is

one which does not change very much over the region in which the likelihood is appreciable and does not assume large values outside that range".

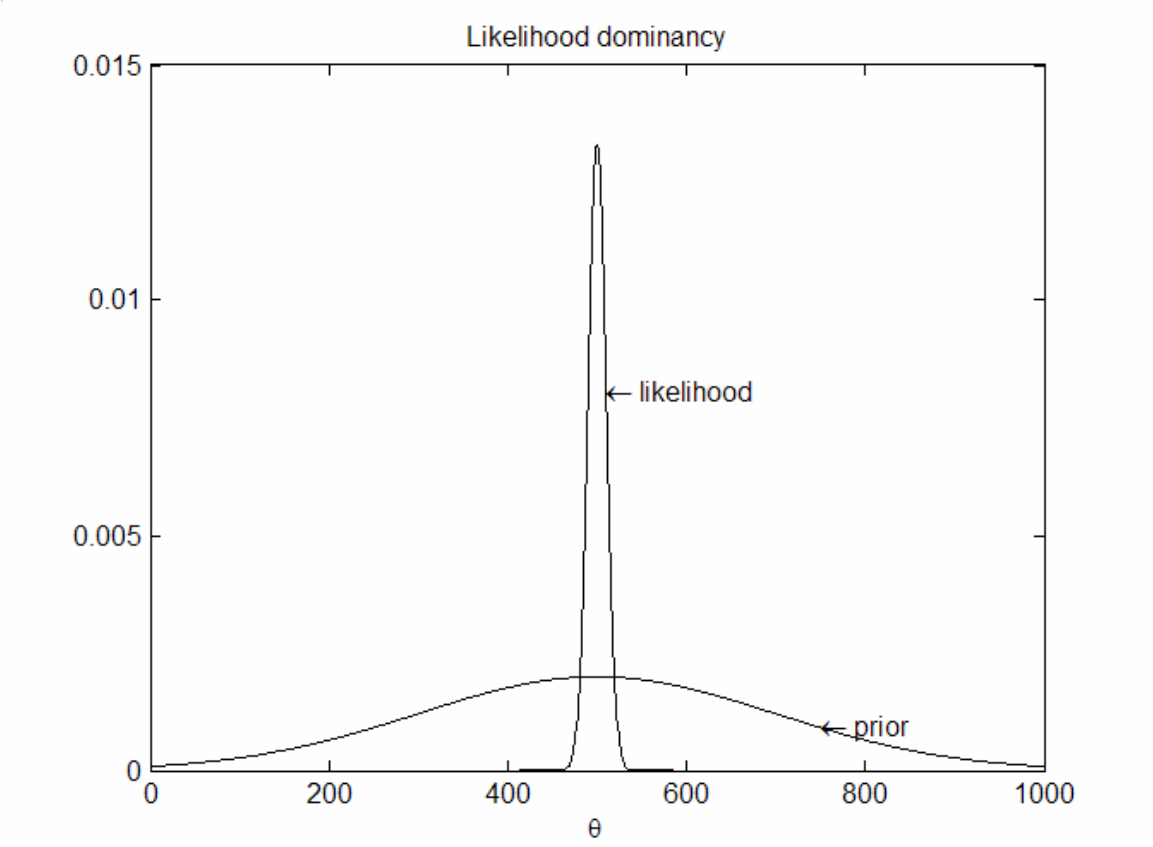


Figure 5.3. Likelihood dominancy (Adapted from Box and Tiao [43])

From Figure 5.3, it is observed that the likelihood dominated prior tends to have a uniform distribution over the range the likelihood is appreciable. This property is called locally uniform. From Equation (5.5), the posterior distribution for a locally uniform prior would be

$$p(\boldsymbol{\theta} \mid \mathbf{y}) = \frac{L(\boldsymbol{\theta} \mid \mathbf{y})}{\int L(\boldsymbol{\theta} \mid \mathbf{y}) d\boldsymbol{\theta}} \tag{5.12}$$

Equation (5.12) means the posterior distribution would be the standardized likelihood. The choice of a prior distribution that is locally uniform means little is known *a prior*. Historically this is called Bayes' postulate. It needs to be pointed out that Bayes' postulate has some fundamental inconsistency in it. This is, however, not a topic that is relevant and should be included this study, which emphasizes practical applications rather than strict theories [43]. Suffice to say, for the purpose of applications, Bayes' postulate can still be accepted. The situation of "little is known *a prior*" suggests the prior distribution is noninformative and thus be called a noninformative prior distribution.

V.4 Noninformative Prior Distribution for Single Variable

Due to its importance in the following sections, noninformative prior distribution is further discussed here. It would be easier to first introduce the single parameter case and then following the same logic, the multi-parameter model can be established and applied to the linear regression model developed in Chapter IV. Here the noninformative prior distribution associated with data translated likelihood is demonstrated through the normal mean and the normal standard deviation.

The Normal mean

Assume $\mathbf{Y} = \{y_1, y_2, ... y_n\}$ is a set of observed outcomes from a Normal distribution $N(\theta, \sigma^2)$ with the standard deviation σ known. The likelihood function in this case, according to Equation (5.7) is

$$L(\theta \mid \sigma, \mathbf{Y}) = c \exp\left\{-\frac{n}{2\sigma^2}(\theta - \bar{y})^2\right\}, \tag{5.13}$$

where c is a normalization factor and $\bar{y}$ is the mean of the observed outcomes.

From Equation (5.13), the likelihood function is graphically represented by a Normal curve centered at $\bar{y}$. The change in the data will only have an impact on the mean $\bar{y}$, which in turn causes a translation of the likelihood on the θ axis as demonstrated in Figure 5.4 on the next page. Therefore, the statement that little is known *a prior* relative to the data would be the same as to say that one value of θ is equally possible as another prior to the acquisition of data. The dash line on the figure represents the "ideal" noninformative prior for θ, which keeps uniform from minus infinity to infinity. It is important to note that the "ideal" prior is just a wishful delusion. A uniform distribution can never exist over an infinite range because its integral over the range has to be "1". Therefore, to say a noninformative prior exists such that it holds for any possible value of θ is mathematically wrong. However, such an "ideal" distribution is completely unnecessary for the practical purpose because most of the applications in the real world have a limited range of θ. The centerline on the figure indicates a "practical"

noninformative prior for θ , which is the graph for a normal distribution and it is a good approximate noninformative prior.

The posterior distribution of the Normal mean is still a normal distribution. Actually from Equation (5.13), the posterior distribution is obtained as

$$p(\theta \mid \sigma, \mathbf{Y}) = (\frac{2\pi\sigma^2}{n}) \exp\left\{-\frac{n}{2\sigma^2}(\theta - \bar{y})^2\right\} \tag{5.14}$$

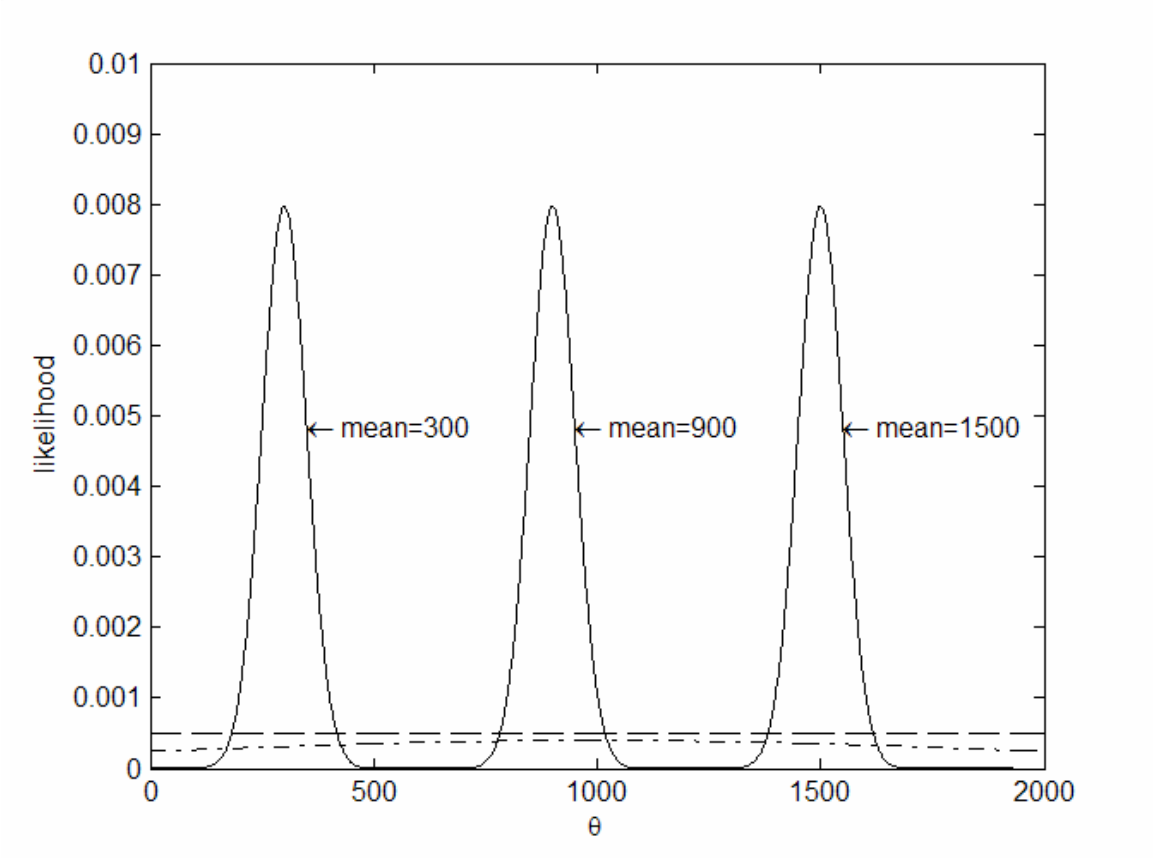


Figure 5.4. The normal mean

The Normal standard deviation

Similar to the normal mean, a noninformative prior can be found for the normal standard deviation. Assume $\mathbf{Y} = \{y_1, y_2, ... y_n\}$ is a set of observed outcomes from a Normal distribution $N(\theta, \sigma^2)$ with the mean θ known. The likelihood function, according to Equation (5.7), is

$$L(\sigma \mid \theta, \mathbf{Y}) = c \exp\left\{-\frac{ns^2}{2\sigma^2}\right\}, \tag{5.15}$$

where $s^2 = \sum_{i=1}^{n} \frac{(y_i - \theta)^2}{n}$.

The likelihood function for different values of *s* is shown in Figure 5.5a. From the figure, it can be seen that the noninformative prior cannot be locally uniform. However, if the logarithm of the standard deviation is considered, the noninformative prior becomes locally uniform again and is shown in Figure 5.5b, where the dash line represents the locally uniform distribution. The posterior distribution does not have an explicit analytical form since the integral does not have analytical solution. However it can be put in the form of gamma-function, as shown in the following equation, where Γ is the gamma function.

$$p(\sigma \mid \theta, \mathbf{Y}) = \frac{(ns^2)^{n/2}}{2^{(n/2)-1}\Gamma(n/2)} \exp\left\{-\frac{ns^2}{2\sigma^2}\right\} \tag{5.16}$$

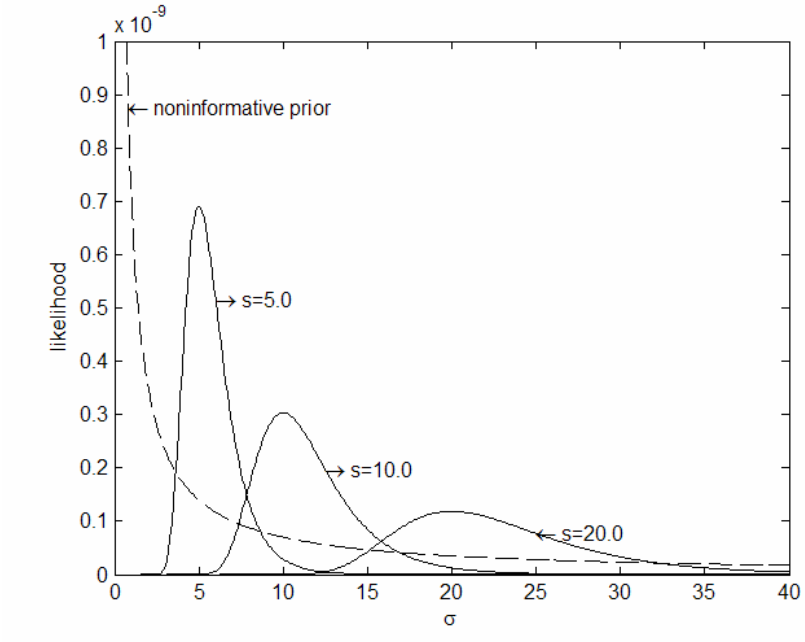


Figure 5.5a. Normal standard deviation (Adapted from Box and Tiao [43])

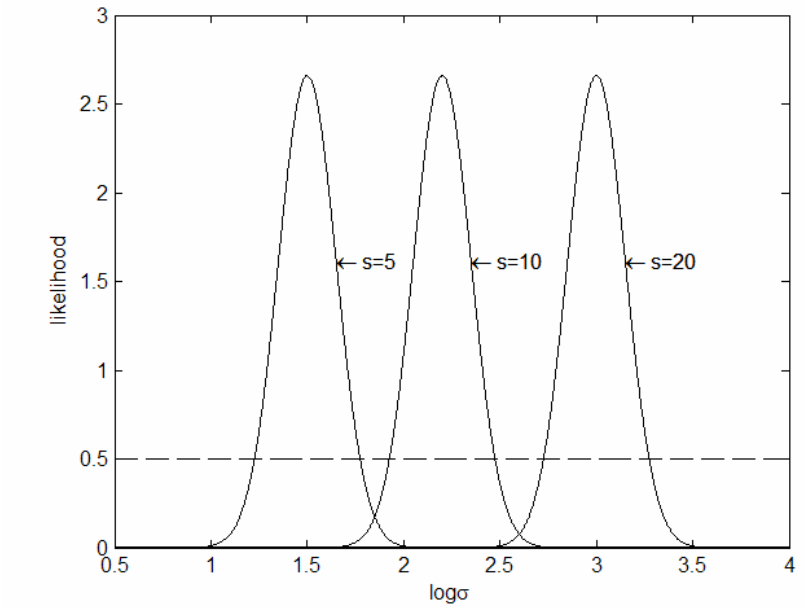


Figure 5.5b. Logarithm of Normal standard deviation (Adapted from Box and Tiao [43])

From the discussion above, it is clear that the requirement for a little-is-known prior distribution can be readily fulfilled in the case of normal mean and normal standard deviation. These examples are important because the model incorporating the linear regression and Bayes' updating rule for multi-variable cases can be built by directly extending these single variable cases.

V.5 Noninformative Priors for Multiple Variables

Multi-variable distribution with σ^2 known

Assume that $\mathbf{Y} = \{y_1, y_2, ... y_m\}$ is a set of statistically independent variables that have normal distribution with the same variance σ^2. And the expected value of $\mathbf{Y}$ is linearly related to *n* parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, ... \theta_n\}$, which means

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\theta}, \tag{5.17}$$

where X is a *m* by *n* constant matrix. Note that Equation (5.17) is actually the linear regression model established in Chapter IV.

If the standard deviation σ is known, the likelihood function is

$$L(\boldsymbol{\theta} \mid \sigma, \mathbf{Y}) = c \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{Y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\theta})\right\} \tag{5.18}$$

To facilitate the discussion, the exponential index can be put in another form. Assume that matrix X is full-ranked. The exponential index except the $-\frac{1}{2\sigma^2}$ is then

$$(\mathbf{Y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{Y}-\mathbf{X}\boldsymbol{\theta})=(\mathbf{Y}-\hat{\mathbf{Y}})^T(\mathbf{Y}-\hat{\mathbf{Y}})+(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})^T\mathbf{X}^T\mathbf{X}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}}), \tag{5.19}$$

where $\hat{\boldsymbol{\theta}}=(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. Note that $\hat{\boldsymbol{\theta}}$ is exactly the coefficients obtained through linear regression in Chapter IV. Now consider that the first term in Equation (5.19) does not involve $\boldsymbol{\theta}$ so that it can be removed from the likelihood function, or alternatively put in the constant c to form a new constant. For convenience, the symbol c is still used here. Then the likelihood function is

$$L(\boldsymbol{\theta}\mid\sigma,\mathbf{Y})=c\exp\left\{-\frac{1}{2\sigma^2}(\hat{\boldsymbol{\theta}}-\boldsymbol{\theta})^T\mathbf{X}^T\mathbf{X}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right\} \tag{5.20}$$

Again, it should be noted that the matrix $\frac{1}{2\sigma^2}\mathbf{X}^T\mathbf{X}$ is the inverse matrix of the covariance matrix mentioned in Chapter IV. For $n=2$, $\sigma=0.03$, $c=1.0$ and $\mathbf{X}=\begin{bmatrix}0.01 & 0.02\\ 0.01 & 0.01\end{bmatrix}$, the contour plots of the likelihood functions with different $\hat{\boldsymbol{\theta}}$ are shown in Figure 5.6. From this figure, the likelihood functions are completely determined except for their locations before the data on $\mathbf{Y}$ becomes available. It is obvious that the likelihood function is determined except for its location. Different data sets would only translate the likelihood function on the (θ_1, θ_2) plane. Therefore a noninformative prior distribution would be a locally uniform distribution that can be expressed as $p(\boldsymbol{\theta}\mid\sigma)\propto c$ where c is a constant. Of course, the "true" locally uniform prior distribution can be a two-dimensional normal distribution with a large standard deviation so that it can be approximately considered to be "flat" over the studied range.

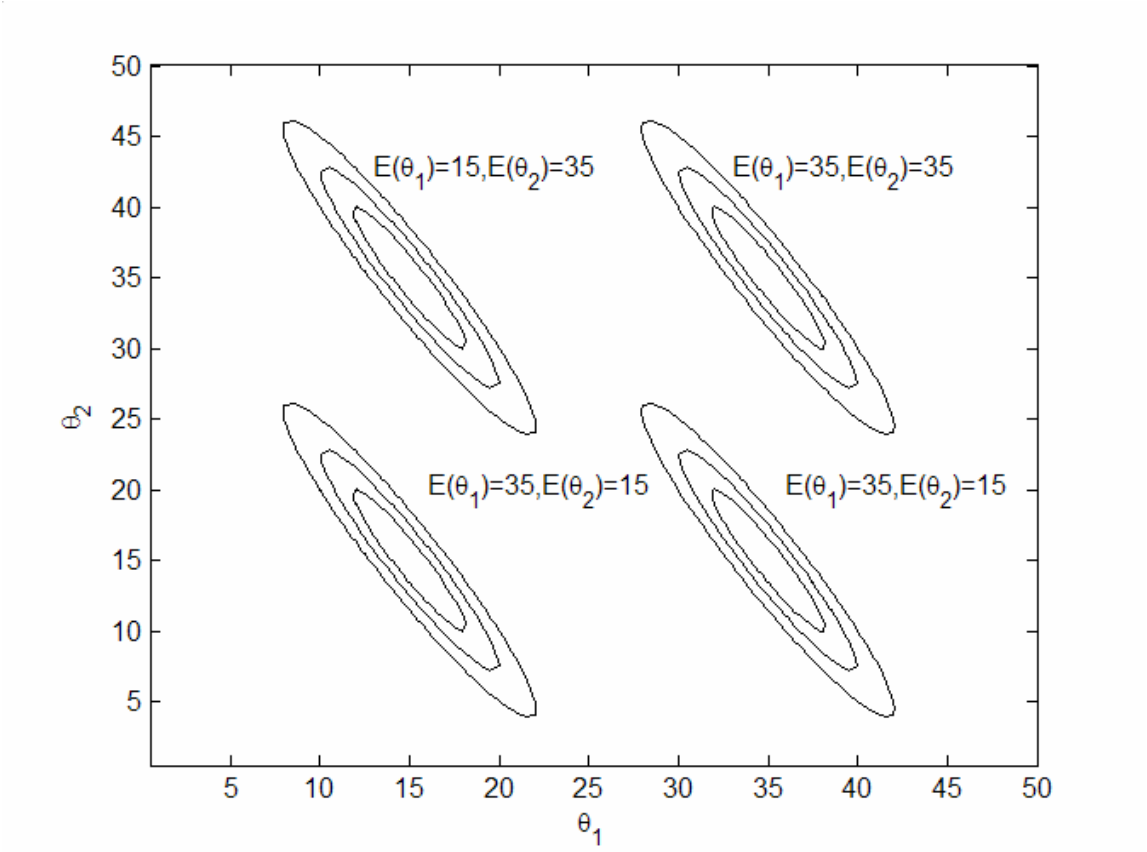


Figure 5.6. Contour plots of likelihood functions for different data sets

From Bayesian Theorem described in Equation (5.4), the posterior distribution of **θ** is

$$p(\boldsymbol{\theta} \mid \sigma, \mathbf{Y}) = \sqrt{\frac{\det(X^T X)}{(2\pi\sigma^2)^k}} \exp\left\{-\frac{1}{2\sigma^2}(\hat{\boldsymbol{\theta}}-\boldsymbol{\theta})^T \mathbf{X}^T\mathbf{X}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right\} \tag{5.21}$$

Equation (5.21) gives a joint normal distribution of multiple variables. The above discussion leads to an important observation. If one is asked to choose a distribution for **θ** without being given any data of **Y**, then it is natural to assume a noninformative prior distribution, or alternatively, a locally uniform distribution. However, once some data on Y becomes available, an initial updating process becomes possible. And the result from

this initial updating process is a joint normal distribution of $\boldsymbol{\theta}$. Keep in mind that the linear regression is also built on the initially available data of **Y.** From here, it is clear that the process to build a linear regression model, as what is described in Chapter IV, is also the process to initially update an assumed locally uniform distribution of the coefficients in the linear regression model. This is how the Bayesian updating rule can be perfectly incorporated with linear regression. This is also the beauty of the methodology proposed in this dissertation.

The objective of studying the Bayesian analysis, however, is not just to verify that the initial updating and the linear regression can be perfectly incorporated. More importantly, the posterior joint normal distribution can be, again, updated following the same methodology if new data is present. In this case, the posterior joint normal distribution is treated as a prior distribution and the likelihood function is derived from the new data. One will see, in the next section, that after a new round of updating, the posterior distribution is, again, a joint normal distribution. Now it is understandable that the updating rule developed here leads to an iterative process that can be repeated as many times as needed. Each time, it included some new information that is not included in the previous model.

However, before the updating rule on a joint normal distribution can be presented, one question can be legitimately asked if one noticed that the previous discussion is based on known standard deviation σ. The question is what would happen if the standard deviation, σ is also unknown.

Multi-variable distribution with σ^2 unknown

It would be easier to first consider a Normal distribution $N(\theta,\sigma^2)$, where both θ and σ are unknown. From the data $\mathbf{Y}=\{y_1, y_2,...y_m\}$, the likelihood function, given the data $\mathbf{Y}$ is

$$L(\theta,\sigma\,|\,\mathbf{Y})\propto\sigma^{-n}\exp\{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\theta)^2\}, \tag{5.22}$$

In Equation (5.22), the exponential index can be written as,

$$\sum_{i=1}^{n}(y_i-\theta)^2=\sum_{i=1}^{n}(y_i-\bar{y})^2+n(\bar{y}-\theta)^2=(n-1)s^2+n(\bar{y}-\theta)^2, \tag{5.23}$$

where $s^2=\sum_{i=1}^{n}\frac{(y_i-\bar{y})^2}{n-1}$. Then the likelihood function is now

$$L(\theta,\sigma\,|\,\mathbf{Y})\propto\sigma^{-n}\exp\{-\frac{(n-1)s^2}{2\sigma^2}-\frac{n(\theta-\bar{y})^2}{2\sigma^2}\} \tag{5.24}$$

Since the likelihood function would be unchanged after multiplying a constant, Equation (5.24) can be further written as

$$L(\theta,\sigma\,|\,\mathbf{Y})\propto\left(\frac{s}{\sigma}\right)^{-n}\exp\{-\frac{(n-1)s^2}{2\sigma^2}-\frac{n(\theta-\bar{y})^2}{2s^2}\left(\frac{s}{\sigma}\right)^2\} \tag{5.25}$$

Apparently the joint distribution of σ and θ represented by Equation (5.25) is not a normal distribution. To seek a noninformative prior distribution, an intuitive way would be to plot the likelihood function. Of course, it can be done mathematically and the procedure is recorded in many statistical textbooks and will not be recorded here.

However, the σ axis should be plotted in the logarithm scale to get a clean picture. To get the logarithm scale, Equation (5.25) is changed to

$$L(\theta,\sigma \mid \mathbf{Y}) \propto \left(\frac{s}{\sigma}\right)^{-n} \exp\{-\frac{n(\theta-\bar{y})^2}{2s^2}\exp[-2(\log\sigma-\log s)\} \\ \times\exp\{-n(\log\sigma-\log s)-(\frac{n-1}{2})\exp[-2(\log\sigma-\log s)]\} \tag{5.26}$$

Figure 5.7 shows the likelihood function in Equation (5.25) with $n = 10$ and different $\bar{y}$ and s. From this figure, when s is fixed, different $\bar{y}$ leads to the relocation of the contour along the θ axis but the shape of the contour is retained, which suggests a noninformative distribution of locally uniform for θ. When $\bar{y}$ is fixed, different s leads to relocation and a scaling of the contour. This suggests a noninformative distribution for σ would not be locally uniform. However, as discussed in V.5.2, the distribution of the logarithm of σ is locally uniform. Therefore a noninformative joint distribution for θ and σ would be locally uniform. This conclusion can be extended to the multivariate cases when θ becomes a vector rather than a scalar.

$$p(\theta,\log\sigma) = c \tag{5.27}$$

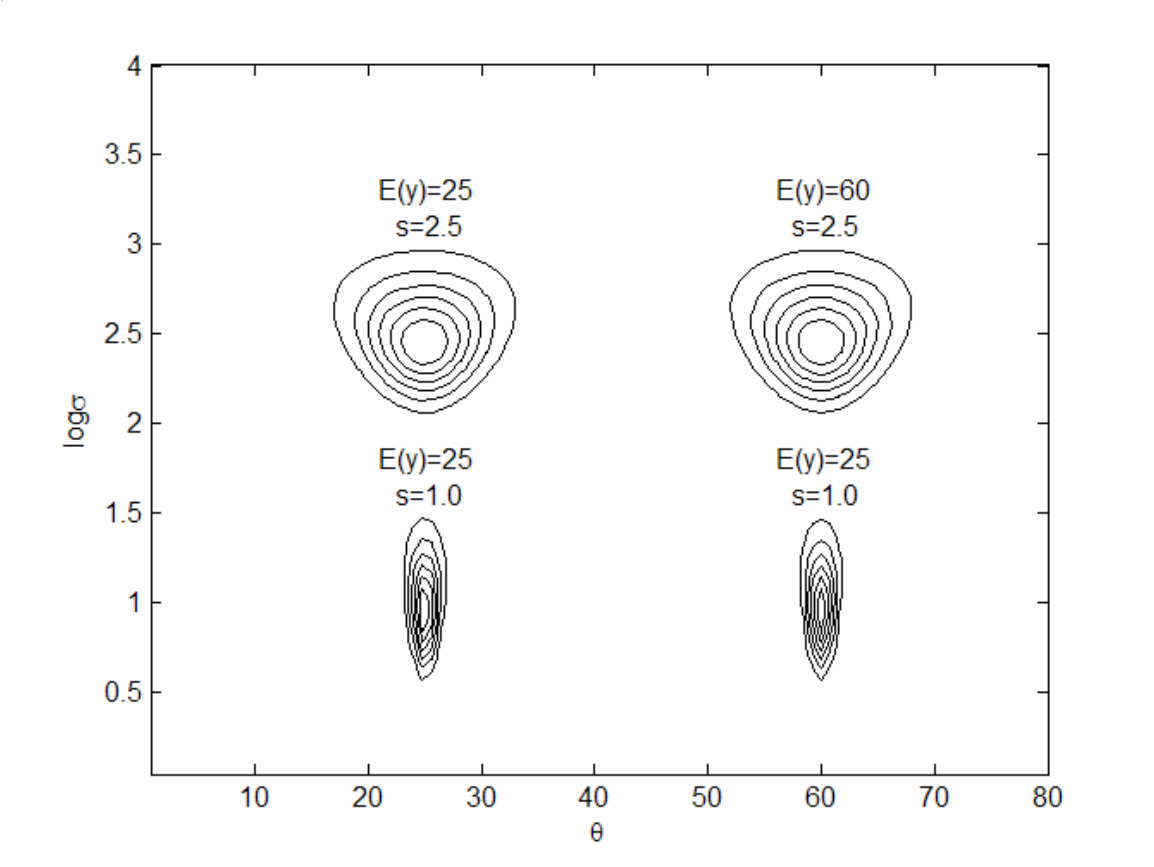


Figure 5.7. Contour plots of likelihood function with both θ and σ unknown

From here, the multi-variable case can be easily obtained. The likelihood function is

$$L(\theta,\sigma\,|\,Y)\propto\left(\frac{s}{\sigma}\right)^{n}\exp\left[-\frac{(n-k)s^{2}}{2\sigma^{2}}-\frac{s^{2}}{2\sigma^{2}}\frac{(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})^{T}\mathbf{X}^{T}\mathbf{X}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})}{s^{2}}\right], \tag{5.28}$$

where $s^{2}=\frac{1}{(n-k)}(\mathbf{Y}-\hat{\mathbf{Y}})^{T}(\mathbf{Y}-\hat{\mathbf{Y}})$. Following the same argument as in the single variable case, the change in $\hat{\boldsymbol{\theta}}$ only changes the location of the likelihood function and

the change in s rescales the likelihood function. Therefore the noninformative prior can be written as

$$p(\boldsymbol{\theta}, \log\sigma) = c \ , \tag{5.29a}$$

or

$$p(\boldsymbol{\theta}, \sigma) = 1/\sigma \tag{5.29b}$$

V.6 Distribution of coefficients in the linear regression model

Now it is appropriate to go back to the linear regression model developed in Chapter IV. Consider the linear model described in Equation (5.15),

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\varepsilon} \tag{5.30}$$

where $\boldsymbol{\varepsilon}$ is the error. Assume that $\boldsymbol{\varepsilon}$ has the multivariate Normal distribution with zero mean and standard deviation σ, i.e., $\boldsymbol{\varepsilon} \sim N(0, \mathbf{I}\sigma^2)$. I is a $n \times 1$ vector that has all entries equal to "1", which means the error terms share the same standard deviation.

From the discussion in the previous section, the joint prior distribution is locally uniform and can be expressed as Equation (5.29b). Therefore the posterior distribution of $\boldsymbol{\theta}$ and σ given $\mathbf{Y}$ is

$$p(\boldsymbol{\theta}, \sigma \mid \mathbf{Y}) = C_0 \frac{1}{\sigma^{n+1}} \exp\left[-\frac{(n-k)s^2}{2\sigma^2} - \frac{s^2}{2\sigma^2} \frac{(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{X}^T \mathbf{X} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})}{s^2} \right] \tag{5.31}$$

where C_0 is a normalization constant.

From here, the marginal distribution of σ^2 is

$$p(\sigma^2 \mid \mathbf{Y}) = (n-k)s^2 \chi^2_{(n-k)} \quad , \tag{5.32}$$

which is the χ-distribution with the number of degree of freedom *n-k*. The mean and variance of the χ-distribution are $(n-k)s^2/(n-k-2)$ and $2(n-k)^2 s^4/((n-k-2)^2(n-k-4))$, respectively [44].

It is very interesting that the variance of the χ-distribution decreases with increasing number of degree of freedom *n-k*. It is apparent that if *n-k* goes to infinity, which means the number of data points is much larger than the number of coefficients in the model, then the probability density function of the χ-distribution will approach to the Dirac delta function [46] located at the mean of the distribution. In this case, the marginal distribution of $\boldsymbol{\theta}$, will approach a distribution described by the following equation.

$$p(\boldsymbol{\theta} \mid \mathbf{Y}) \to C_0 \frac{1}{\sigma^{n+1}} \exp\left[-\frac{(n-k)s^2}{2\sigma^2} - \frac{s^2}{2\sigma^2} \frac{(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{X}^T \mathbf{X} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})}{s^2} \right], \tag{5.33}$$

with $\sigma^2 = (n-k)s^2/(n-k-2)$, when $n-k \to \infty$.

Consider that σ^2 is now a constant, the distribution described in Equation (5.33) can be rewritten as

$$p(\boldsymbol{\theta} \mid \mathbf{Y}) \to C_1 \exp\left[-\frac{(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{X}^T \mathbf{X} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})}{2\sigma^2} \right] \tag{5.34}$$

This is a normal distribution centered at $\hat{\boldsymbol{\theta}}$, with a variance of $\sigma^2 = (n-k)s^2/(n-k-2)$. Again C_1 is a normalization constant. This is a very

important observation. First, it states that the conditional distribution of the coefficient **θ** is approximately a normal distribution when the number of degree of freedom is large enough. Actually in their book, Box and Tiao's [43] stated that "in particular, for large number of degrees of freedom, the distribution of **θ** will be very nearly normal because the conditional normal distribution of **θ** on condition of σ and **Y** will be averaged only over a narrow 'weight' distribution". Mathematically, it has to be infinity. However, in practice, infinity can never be reached and it is not necessary to go to infinity. When the number of degree of freedom is reasonably large, the conclusion holds. For the regression model in this dissertation, $s = 0.2293$, $n\text{-}k = 35$, the variance of the distribution is *1.7e-4*, which can be regarded as small so it is reasonable to consider the marginal distribution of **θ** as a normal distribution.

Secondly, it is a very nice property that the initial updating from the locally uniform distribution leads to a normal distribution. The reason is that only a normal distribution can be readily updated with new incoming data and leads to a new normal distribution. This is very important and it is the beauty of the entire methodology because it basically says the updating rule can be repeatedly applied to the model without changing the type of distribution, but only the mean and variance of the distribution. An iteration process is thus possible. Following is the detailed discussion.

The distribution of the coefficient **θ,** in the baseline linear regression model, which is based on the 43 selected software, can be now written as

$$p(\boldsymbol{\theta} \mid \mathbf{Y}) = \frac{1}{(\sqrt{2\pi})^n \sqrt{\det(\Sigma)}} \exp\left[-\frac{(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \Sigma^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})}{2} \right] \tag{5.35}$$

with $\Sigma = (n-k)s^2(\mathbf{X}^T\mathbf{X})^{-1}/(n-k-2)$ and $\hat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$, both of which can be calculated from the collected data. And this distribution is now treated as a prior distribution for any future updating.

Now data from a new software become available. Assume the data can be represented by $(Y_{new}, \mathbf{X}_{new})$. And Y_{new} is the final score of the new software and $\mathbf{X}_{new}$ is the scores for each item. The likelihood function of the new data, which is actually the likelihood of the error from the regression model, is

$$L(\boldsymbol{\theta} \mid y_{new}) \propto \exp\left[-\frac{1}{2\sigma^2}(y_{new} - \boldsymbol{\theta}^T \bar{\mathbf{X}}_{new})^2\right] \tag{5.36}$$

Apply the Bayesian theorem again, the posterior distribution is then

$$p(\boldsymbol{\theta} \mid \mathbf{Y})_{updated} = A_0 \exp\left[-\frac{(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \Sigma^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})}{2} - \frac{1}{2\sigma^2}(y_{new} - \boldsymbol{\theta}^T \bar{\mathbf{X}}_{new})^2\right] \tag{5.37}$$

where A_0 is a normalization constant. While the distribution described in Equation (5.37) looks complicated, it is actually a joint normal distribution, again. The reason is that the characteristic of a joint normal distribution is the quadratic form in terms of $\boldsymbol{\theta}$. In Equation (5.37), $\frac{(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \Sigma^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})}{2}$ is a quadratic form and $(y_{new} - \boldsymbol{\theta}^T \bar{\mathbf{X}}_{new})^2$ is a quadratic form again. The sum of two quadratic forms can always be written as a new quadratic form plus a constant. Since the constant can be always moved into the normalization constant A_0, it means the distribution described in Equation (5.37) has a quadratic form in its exponential index and thus is a joint normal distribution. This new distribution turns out to be

$$p(\boldsymbol{\theta} \mid \mathbf{Y})_{updated} = \frac{1}{(\sqrt{2\pi})^n \sqrt{\det(\sum)}} \exp\left[-\frac{(\boldsymbol{\theta} - \boldsymbol{\theta}_{updated})^T \sum_{updated}^{-1} (\boldsymbol{\theta} - \boldsymbol{\theta}_{updated})}{2} \right] \tag{5.38}$$

with

$$\hat{\boldsymbol{\theta}}_{updated} = (\Sigma^{-1} + \frac{1}{\sigma^2} \bar{\mathbf{X}}_{new} \bar{\mathbf{X}}_{new}^T)^{-1} (\hat{\boldsymbol{\theta}}^T \boldsymbol{\Sigma}^{-1} + \frac{y_{new}}{\sigma^2} \bar{\mathbf{X}}_{new}^T) \tag{5.39}$$

Equation (5.39) is the updating rule being sought. It relates the old mean of the coefficients $\boldsymbol{\theta}$ and the new updated mean of the coefficients $\hat{\boldsymbol{\theta}}_{updated}$.

Still the standard deviation and covariance matrix need to be updated. To update the standard deviation, s and $\eta=n-k$ must be updated first. This is easy.

$$s_{updated} = \sqrt{s^2 + \frac{1}{\eta}(y_{new} - \bar{y}_{new})} \tag{5.40}$$

where $\bar{y}_{new}$ is the estimated total score from the regression model for the new software.

$$\eta_{updated} = \eta + 1 \tag{5.41}$$

To update $\boldsymbol{\Sigma}$, $\mathbf{X}^T\mathbf{X}$ must be updated first.

$$(\mathbf{X}^T\mathbf{X})_{updated} = \mathbf{X}^T\mathbf{X} + \mathbf{X}_{new}^T\mathbf{X}_{new} \tag{5.42}$$

$$\sum_{updated} = (n-k)s^2 (\mathbf{X}^T\mathbf{X})_{updated}^{-1} / (n-k-2) \tag{5.43}$$

Equation (5.39) through (5.43) establishes the updating rules. It is seen that the original data used to build the regression model is not necessarily in the updating process. All that is needed is only the statistical parameters in the regression model, represented by $\hat{\boldsymbol{\theta}}$, $\boldsymbol{\Sigma}$ and σ. This is a very nice property since it reduced the amount of necessary information. Remember that $\mathbf{X}^T$ is a 8×43 matrix and its number of columns will

increase with each updating performance but $\mathbf{X}^T\mathbf{X}$ is always 8×8 no matter how many times the model is updated. There is no need to maintain a long list of data used for the previous updating. Therefore it can be concluded that the proposed updating rule provides a convenient and easy-to-implement method.

CHAPTER VI

CONCLUSION AND FUTURE WORK

The need for an objective way to assess open source software has become urgent as the number of open source software in use becomes increasingly large. The basic idea is to provide an objective approach through which the open source software can be appropriately evaluated and chosen by average users when they face a huge number of choices. This dissertation has proposed a quantitative method to evaluate open source software based on linear regression and Bayesian statistical analysis.

The linear regression model aims at relating the total score of an open source software with a number of evaluation metrics. Each of these metrics reflects the property or performance of the software in a particular field. Some of the metrics are commonly used by existing methods. In this dissertation, there are totally seven metrics chosen to represent the strength of an open source software in a certain field. These metrics are popularity represented by page rank, activities, quality represented by the number of bugs, support represented by the number of messages, documentation represented by the number of documentation lines and the number of books, and finally integration represented by the number of dependent projects. The total score and the metrics of a particular software can be obtained on popular open source software websites. Given the large number of users on these sites, the total score and the metrics can be considered to correctly reflect the performance of the software.

To build a linear regression model, the collected data need to satisfy some assumptions. These assumptions ensure that the model gives reasonable prediction for any future application. These assumptions refer to “constant variance”, “independence” and “normality”. The data collected for the open source software are put to test against these assumptions. It is found that they satisfy these assumptions very well. Therefore the linear regression model is plausible. The important issue of outlier removal in linear regression model is also presented in this dissertation. The sifting process and the model development are intrinsically intertwined so it is actually an integrated trial and error process. Through this process, a total number of 43 software are finally chosen to build the baseline model. The model is built through SPSS, the popular statistical kit. It is found that the coefficients obtained are reasonable. The model is then applied to a number of new software to test its accuracy. It is found that the error between the predicted score and the user review score is within 5%. It shall be understood that this is not to say the model will give close results for any software in the vast selection of open source software. To introduce the influence of new software that is not included in the 43 selected software, a statistical updating process needs to be developed.

The updating process is based on Bayesian analysis. In the Bayesian analysis, the prior distribution of a random variable and its posterior distribution are related by the likelihood function, which reflects experiment results or simply, data. The coefficients in the linear regression model are then treated as random variables that have a certain distribution. Before the regression model is built, it is assumed that all coefficients have noninformative prior distribution. After the model is built the error is assumed to have

normal distribution, the posterior distribution of the coefficients is actually T-distribution. However, if the number of degrees of freedom, which is the number of data points or the number of software used to build regression model subtracted by the number of coefficients, is reasonably large. The distribution approaches a normal distribution. The mean and the standard deviation can be found through the data used to build the regression model.

The normal distribution of the coefficients can be further updated by future incoming data. The posterior distribution or updated distribution is found to be normal again. Thus an iteration process is set up and it can be repeated as many times as needed. Formulas are given for updating the mean and standard deviation of the coefficients. A nice feature of this process is that it does not require previous data on which the current model is built. It only needs the statistical parameters of the current model. Thus the amount of information is reduced and it makes this method easy to implement.

There are several things can be done as future work to improve the model in this dissertation. First, some of the evaluation metrics can be reconsidered. For instance, the number of books, which represents the document metric, is zero for most of the software. This polarization reduces the linearity of the intrinsic model thus leads to errors in the linear regression. Secondly, on the regression side, other type of regression could be tried. As indicated by the standard deviation and 95% confidence interval, the linear regression approach is probably not the best way to relate the total score and the metrics. Linear regression model is widely used just because it is easy to implement. This is exactly why today's most famous method in open source software evaluation also

uses linear regression. From here, it can be seen that the type of model and the appropriate choice of metrics are two interactive factors. The relaxation on one leads to tightness on the other. That is, if the metrics are allowed to be chosen more freely, the model has to be more complicated than a linear model. On the other hand, if the metrics are chosen more carefully to accommodate more intrinsic linearity, the linear model will be sufficient.

Finally it needs to be pointed out that the methodology developed in this dissertation may have wide range of applications given the popularity of linear regression in the science and engineering fields. Developing a new method is exactly the primary contribution and purpose of this dissertation.

REFERENCES

[1] W.S. Humphrey, "Characterizing the Software Process: A Maturity Framework", in *IEEE Software*, vol. 5, no. 2, pp. 73-79, March 1988

[2] W.S. Humphrey, *Managing the Software Process*, Addison-Wesley, 1989

[3] Carnegie Mellon University Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995

[4] M. Cusumano, C. Shirky, J. Feller, B. Fitgerald, S. Hissam, and K. Lakhani, *Perspectives on Free and Open Source Software*, MIT Press: Cambridge, MA, 2005

[5] Government of State of California Performance Review, http://cpr.ca.gov/report/cprrpt/issrec/stops/it/so10.htm

[6] Free and Open Source Software Overview and Preliminary Guidelines for the Government of Canada, http://www.tbs-sct.gc.ca/fap-paf/oss-ll/foss-llo/foss-llo17_e.asp

[7] A. Wasserman, M. Pal, and C. Chan, "The Business Readiness Rating Model: An Evaluation Framework for Open Source", *EFOSS - Evaluation Framework for Open Source Software Workshop*, June, 2006

[8] D.A. Wheeler, "How to Evaluate Open Source Software / Free Software (OSS/FS) Programs", http://www.dwheeler.com/oss_fs_eval.html

[9] B. Golden, *Succeeding with Open Source*, Addison Wesley, 2005

[10] K. van den Berg, “Finding Open Options, An Open Source Software Evaluation Model with a Case Study on Course Management Systems”, Master Thesis, Tilburg University, August 2005

[11] M. Surman and J. Diceman, “Choosing Open Source, A Decision Guide for Civil Society Organizations”, http://www.itrainonline.org, January 2004

[12] A. Cau, G. Concas, and M. Marchesi, “Extending OpenBRR with Automated Metrics to Measure Object Oriented Open Source Project Success”, *The Second International Conference on Open Source Systems*, Italy, June, 2006

[13] K. Crowston, H. Annabi, and J. Howison, “Defining Open Source Software Project Success”, *Proceedings of International Conference on Information Systems* (ICIS), pp. 327-340, 2003.

[14] D. Weiss, “Measuring success of open source projects using web search engines”, In OSS2005: *Proceedings of The First International Conference on Open Source Systems*, pp. 93-99, 2005.

[15] “Top Tips for Selecting Open Source Software”, http://www.oss-watch.ac.uk/resources/tips.xml

[16] M.C. Paulk, *Key Practices of the Capability Maturity Model*. Reading, MA:Addison-Wesley, 1993.

[17] M. Chrissis, M. Konrad, and S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement* (2nd Edition), Addison-Wesley Professional, 2003

[18] M. Kulpa and K. Johnson, *Interpreting the CMMI : A Process Improvement Approach*, Auerbach Publications, 2003

[19] R. Turner, and A. Jain, "Agile Meets CMMI: Culture Clash or Common Cause?", *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe*, pp. 153–165, 2002

[20] J.R. Nawrocki, M. Jasinski, B. Walter, and A. Wojciechowski, "Extreme Programming Modified: Embrace Requirements Engineering Practices", *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, pp. 303-310, 2002

[21] B. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[22] B.W. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", *Annals of Software Engineering*, vol. 1, pp. 57-94, 1995

[23] B. Clark, S. Chulani, and B. Boehm , "Calibration Results of COCOMOII.1997", *International Conference on Software Engineering*, April 1998.

[24] S. Chulani, B. Boehm, and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models", *IEEE Transaction on Software Engineering*, vol. 25, pp. 573-583, 1999

[25] D. Simmons, N. Ellis, H. Fujihara, and W. Kuo, *Software Measurement—A Visualization Tool Kit for Process Control and Process Improvement*, Upper Saddle River, NJ: Prentice-Hall, 1998.

[26] International Function Point Users Group (IFPUG), *Function Point Counting Practices Manual*, Release 4.0, 1994.

[27] R.D. Banker, R.J. Kauffman and R. Kumar, “An empirical test of object-based output measurement metrics in a computer aided software engineering (CASE) environment”, *Journal of Management Information Systems*, vo1. 8, No. 3, 1992.

[28] S. Chulani, B. Boehm, and B. Steece, "Calibrating Software Cost Models Using Bayesian Analysis," *Proceedings 1999 ISPA/SCEA Conference*, 1999

[29] A. Gelman, J. B. Carlin, H. S. Stern and D. B. Rubin, *Bayesian Data Analysis*, Second Edition (Texts in Statistical Science), Chapman & Hall/CRC, 2003

[30] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer, 2003

[31] CapGemini’s Open Source Software Maturity Model, http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf

[32] D. German and A. Mockus, “Automating the Measurement of Open Source Projects”, *Workshop on Open Source Software Engineering* (in ICSE03), Portland, OR, May 2003.

[33] Erenkrantz, J.R. and R.N. Taylor, "Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community", *Proceedings of 1st Workshop on Open Source in an Industrial Context*. Oct, 2003. Anaheim, CA

[34] L. Page, S. Brin, R. Motwani, and T. Winograd, "The Pagerank Citation Ranking: Bringing Order to the Web", Technical report, Stanford Digital Library Technologies Project, 1998.

[35] Google Technology, http://www.google.com/technology/

[36] B. Bowerman and R. O'Connell, *Linear Statistical Models-An Applied Approach*, Second Edition, PWS-KENT Publishing Company, Boston, 1990

[37] D.C. Montgomery, E.A. Peck, C.G. Vining, *Introduction to Linear Regression Analysis*, 4th Edition, John Wiley & Sons Inc, NJ, 2006.

[38] I. Pardoe, *Applied Regression Modeling*, John Wiley & Sons Inc, 2006.

[39] J. Neter, W. Wasserman, M. Kutner, *Applied Linear Regression Models*, Richard D. Irwin Inc, Homewood, IL, 1983.

[40] M.A. Golberg, H.A.Cho, *Introduction to Regression Analysis*, WIT Press, Southampton, UK, 2004.

[41] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences* , 2nd Edition, Lawrence Erlbaum Associates , Hillsdale, NJ, 1988

[42] S. James Press, *Bayesian Statistics: Principles, Models, and Applications*, John Wiley & Sons Inc, 1989.

[43] G.E.P. Box and G.C. Tiao, *Bayesian Inference Statistical Analysis*, John Wiley & Sons, 2003

[44] R.A. Fisher. "On the Mathematical Foundations of Theoretical Statistics". *Philosophical Transactions of the Royal Society*, A, 222:309-368 (1922).

[45] P.M. Lee, *Bayesian Statistics: An Introduction*, Third Edition, A Hodder Arnold Publication, New York, 2004

[46] R. Bracewell, *The Fourier Transform and Its Applications,* Third Edition, McGraw-Hill, 1999

APPENDIX A

43 OPEN SOURCE SOFTWARE DATA

	User reviews	Page Rank (out of 10)	activities/month/TLOC	#BUG/NUM_MSG	#MSG/TLOC	DOC/TLOC	BOOK	Number of projects depending on this one
MPlayer	9.1	7	3.72E-03	2.06E-02	0.201671	23.11254	0	53
Blitz JavaSpaces	8.61	6	0.11225	0.545455	0.002004	30.51465	0	0
Inkscape	8.73	7	0.098146	0.122622	0.062283	1.89709	1	0
Kannel WAP and SMS Gateway	8.92	6	0.067198	0.002	0.026282	24.72453	0	0
MoinMoin	8.77	8	0.096116	0.022244	0.077614	11.70745	0	0
COID - C++ Networking Code Generator	8.4	3	0.006116	0.470588	0.000195	8.543872	0	0
SynCE - KDE	8.38	5	0.006758	0.052527	0.095468	9.461633	0	0
conexus I/O Library	8.15	4	0.053483	1	6.32E-05	18.96214	0	1
Freenet	7.8	6	0.015243	0.438169	0.109582	14.35126	0	0
mICQ	8.12	5	0.065155	0.00655	0.036559	33.76527	0	1

naim	8.54	6	0.140081	0.005938	0.058974	41.07862	0	0
rssowl	8.53	8	0.011067	0.332335	0.012321	55.08964	0	0
Licq	8.52	5	0.246419	0.015528	0.133179	16.78769	0	0
Interverse	8.5	4	0.030904	0.103175	0.014602	3.476649	0	0
simpleAIM	8.47	5	0.096819	0.20339	0.019993	16.94341	0	0
afd	8.43	5	0.009459	0.08871	0.000505	12.84566	0	0
GNU virtual private ethernet	8.39	5	0.049799	0	0.002014	18.41462	0	0
Platform Independent Petri Net Editor	8.34	5	0.003052	0.590909	0.001007	60.42849	0	0
Liferea	8.26	8	0.035485	0.155101	0.059863	13.14833	0	0
GNU Gadu	8.19	5	0.125705	0.065518	0.070056	3.716409	0	0
MyDNS	8.83	6	0.024345	0.005568	0.068015	49.04188	0	0
Druid	8.79	6	0.030119	0.162413	0.040311	4.676394	0	0
Hierarchical NoteBook	8.75	6	0.061532	0.048193	0.037631	24.39246	0	0
SiteBar	8.68	7	0.020779	0.029543	0.061124	23.72236	0	0
Tellico	8.68	5	0.014596	0.128378	0.016129	21.06037	0	0
OTRS	8.66	6	0.17658	0.065225	3.082743	8.63707	0	0
knoda	8.5	5	0.006964	0.126662	0.025064	33.76304	0	0
OpenLDAP	8.49	9	0.014726	0.204261	0.002742	12.50034	0	26

Firebird .NET Data Provider	8.46	6	0.136747	0.067976	0.177125	46.81692	0	0
building object network databases	8.45	6	0.229579	0	0.000286	1.504804	0	0
csvtosql	8.4	4	0.093738	0.371429	0.004374	13.87327	0	0
CrisoftRicette	8.39	4	0.170343	0.056604	0.009028	13.88297	0	0
YAZ	8.33	6	0.004609	0.142395	0.012455	19.95244	0	7
PHP Generic Access Control List	8.28	6	0.040571	0.122628	0.010587	11.26704	0	0
DWI -- Data With Interaction	8.19	5	0.012072	0.02381	0.002366	31.54752	0	0
jSyncManager	8.19	5	0.033077	0.197183	0.020667	37.69466	0	0
imgSeek	8.1	6	0.090678	0.297071	0.005889	22.59567	0	0
FreeTDS	8.09	6	0.039792	0.113381	0.239605	8.767136	0	0
GTKtalog	7.99	5	0.019688	0.583333	0.001238	6.29093	0	0
Customer-Touch CRM	7.97	4	0.046185	0.615385	0.001801	2.493996	0	0
Xplanet	8.78	7	0.015818	0.13	0.003285	26.48006	0	0
synergy2	8.69	6	0.204643	0.87892	0.019338	11.2984	0	0

rxvt-unicode	8.66	5	0.029635	0.189474	0.006039	27.60297	0	0

VITA

Dongmin Zhang was born in Beijing, P.R. China. She received her Bachelor of Science and Master of Science from Tsinghua University, Beijing, P.R. China. Both majors were in automotive engineering. She also received her Master of Science in computer science from the University of Nebraska at Lincoln with research focus on Image Processing in December 2001. Since January 2002, Dongmin has been a Ph.D student at Texas A&M University, from where she received her Doctor of Philosophy degree in computer science in August 2007. Her permanent address is Chao Wai Hong Miao 13-1-502, Beijing, P.R. China, 100026.