On Measurement and Understanding of Software Development Processes

Ping Zhang
Data Analysis Research Department
Avaya Labs
233 Mt Airy Rd., Basking Ridge, NJ 07920

Audris Mockus
Software Technology Research Department
Avaya Labs
233 Mt Airy Rd., Basking Ridge, NJ 07920

Abstract

Software systems are entities that change constantly throughout their lifetime. Understanding the relationship between different types of changes and the effect of these changes on project outcome is the key problem in software engineering. Primary software change activities involve adding new features and fixing defects. Our fundamental premise is that defects discovered and fixed during development are caused by implementation of new features. Conceptually, this allows us to decompose a software development project into many tiny sub-cycles, one for each new feature. We show that statistical inference under the proposed model is equivalent to the inference problem under a general mixture model with truncated data. We apply the model and the corresponding inference methodology to several historical projects and use it to predict the outcome of a recent project. Our model presents a novel unified framework to investigate and predict effort, schedule, and defects of a software project. The results of applying the model confirm a fundamental relationship between the new feature and defect repair changes and demonstrate its predictive properties.

Key Words and Phrases: Jackknife Method, Maximum Likelihood, Nonhomogeneous Poisson Process, Prediction, Software Changes, Software Engineering.

1 Introduction

The software industry is unlike any manufacturing industry in that its products are abstract conceptual structures described in programming languages and realized on various computer hardwares. Despite all the advances in computer science and software engineering, human intelligence is still the most important factor in software productivity. In fact, some caution that when it comes to software development, it is unrealistic to expect breakthroughs that can lead to productivity gains in orders of magnitude similar to the computer hardware industry (Brooks 1995). Adding to this skepticism is the general consensus that software projects are difficult to plan and manage. It is more than just individuals writing programs. These programs will have to communicate to one another and the complexity generated by such interactions can be overwhelming even for small projects. Therefore, understanding the software development is a formidable research challenge.

In simple terms, the tasks facing a software development organization can be briefly described as follows: a software project starts with a set of requirements that are later translated into architecture design and specifications. Often, the goal is to enhance an existing software product. The work is then partitioned and assigned to developers for low-level design and implementation. Modifications to the software lead to defects that occur either because the new functionality is not implemented correctly or because the existing functionality is exercised in new ways. A large amount of effort is then spent on finding and fixing software defects found through development integration and system test. When everything is deemed satisfactory, the product is packaged and made available to its intended customers and further defects are discoverd by customers in the "field." The cycle then repeats itself with a new release of the software. What makes software engineering difficult is that changes to a software system can be planned or incidental. Adding new features to a software product, for example, is planned activity whereas repairing defects is incidental. The hidden truth in software development is that not only is a software product constantly growing in size and complexity, an overwhelming proportion of such growth is actually caused by incidental changes to the software. This leads to an unprecedented level of uncertainty in software development that is manifested by the wide spread opinion among software professionals that accurate planning of a software project (i.e., schedule, cost, and quality) is extremely difficult, if not impossible.

One way to predict the future is to learn from the past. Indeed, with all the advances in technology and management theory, project estimation is still crucially dependent on experience. The popular COCOMO model (Boehm 1981), although appearing to be data driven, actually depends critically on expert opinions. On the other hand, software organizations routinely collect large amount of data for various purposes. There are project documents that describe what is to be done. Software version control systems keep track of what has been changed to the code base (e.g., which lines of what program files are changed by whom at what time). Change management systems keep records on why changes are initiated (e.g., what errors are identified by whom at what time, and how the error was repaired). Building and testing tools also generate their own data. Finally, the field support organization collects data on customer problems when using the product. What can we learn from these data? More specifically, can we extract useful information from these data that facilitates more objective and better estimation of project status? Here the opportunity for statisticians to contribute is wide open. Our main goal in this paper is to address some of the above issues by appling probability models to describe some of the inherent uncertainties in software processes. Moreover, we also address statistical issues when fitting these models to real data.

Traditional software development follows a sequential "waterfall" model. One starts with a design, which is followed by implementation, testing, and field support. Each step starts when the previous step is finished. Such development models are conceptually simple, easy to manage, but highly inefficient. By contrast, a spiral model means that these sequential steps overlap, are performed several times during the project, and on a smaller scale. The eXtreme programming (XP) technique (Beck 2000) is an application of the spiral model by pushing the concept to its limit. The reality in most modern development environment is somewhere in between. One does not wait until the end to debug a nearly finished system. Neither do we build and test unfinished systems everyday. Regardless of the development model adopted by the organization, one thing remains the same: software development consists of small changes to the source code that are either planned or incidental. A waterfall model is one with most of the planned activities occurring early in the project. In a spiral model, the two types of activities are spread more evenly during the project.

From a statistical perspective, the main objective of analyzing data is to understand uncertainty and to validate scientific hypotheses. In our case, the uncertainty lies in the identification of programming or design errors and the amount of effort required to fix these errors. In other words, we assume that errors are generated randomly following an unknown stochastic mechanism that can be quantified through statistical inference. To further narrow down this uncertainty, we postulate that the root cause of all errors (i.e., incidental changes to the software) is a previous change that is part of the planned activities. Put simply, we assume that all defects are caused by the introduction of new features. Such an assumption, which is reasonably close to reality, will be adequate to allow a whole range of probability models and the corresponding statistical inference methods to be used to tackle the problem. In this paper, we shall explore one such possibilities, namely the use of point process models to describe the defect repair process.

The rest of the paper is organized as follows: In Section 2, we describe the kind of data that is typically available in software development. This is followed by Section 3, where we introduce a Poisson process model for the software development process based on the assumption that defects are caused by implementing features. The next two Sections are focused on issues related to statistical inference and prediction under the Poisson model. The models and the associated inference methods are then applied to some real software project data in Section 6. We close the paper in Section 7 with some discussions on potential future extensions of this work.

2 Software Change Data

Two types of data are used in this paper. Both types of data are automatically collected by configuration management tools routinely used in software projects. First, all changes against the source code are recorded by a version control system. This can be a public domain tool such as the Concurrent Versions System (CVS, see http://www.cvshome.org), which is popular among open source projects, or any of the numerous commercial tools that are available on the market. Most of these tools are various extensions of the SCCS (Rochkind 1975). Among the basic functions, the version control system will keep track of changes made to individual program files (e.g., lines added and removed, developer who made the change, etc), changes made to the file system structure (branching and partitioning), and more importantly the global information that allows the extraction of previous versions at file or system level. In short, a version control system allows one to reconstruct snap shots of the code base at any previous point of time, and doing so in an efficient and user friendly way.

The second data source is the change management system. In professional software development, any changes to the source code need a request typically called modification request (MR) or problem request (PR). The change management system handling these requests will often record the reason for the change (e.g., new feature vs fix), the problem or new feature description, the severity level, and other information. MRs that are not opened by a developer to make a change are reviewed by a team and assigned to a developer who is charged with the responsibility of diagnosing and fixing the problem. Later during the process, the developer will check out program files and edit them. The modified files are then tested and submitted to the code base. From that point on, integration and system tests will be performed. Errors generated by such tests will lead to new MRs. An MR represents the basic work unit in software development.

Many fields in the MR database are entered manually. Although in most cases there are project procedures for entering data, there are not always incentives for developers to follow such procedures except when they can clearly support their own work. Consequently, not all the fields in the MR database are reliable. For example, MRs are often classified and prioritized based on the perceived importance of the problems to be fixed. Unfortunately, the priorities often change in the course of time.

The real value of software change data is that human activities that are essential for the operation of the project are directly linked to the measurable output, i.e., changes made to the source code. The analysis of software change data, as described above, has recently been explored to test some of the well known qualitative predictions in software engineering (for example, Eick et al., 2001).

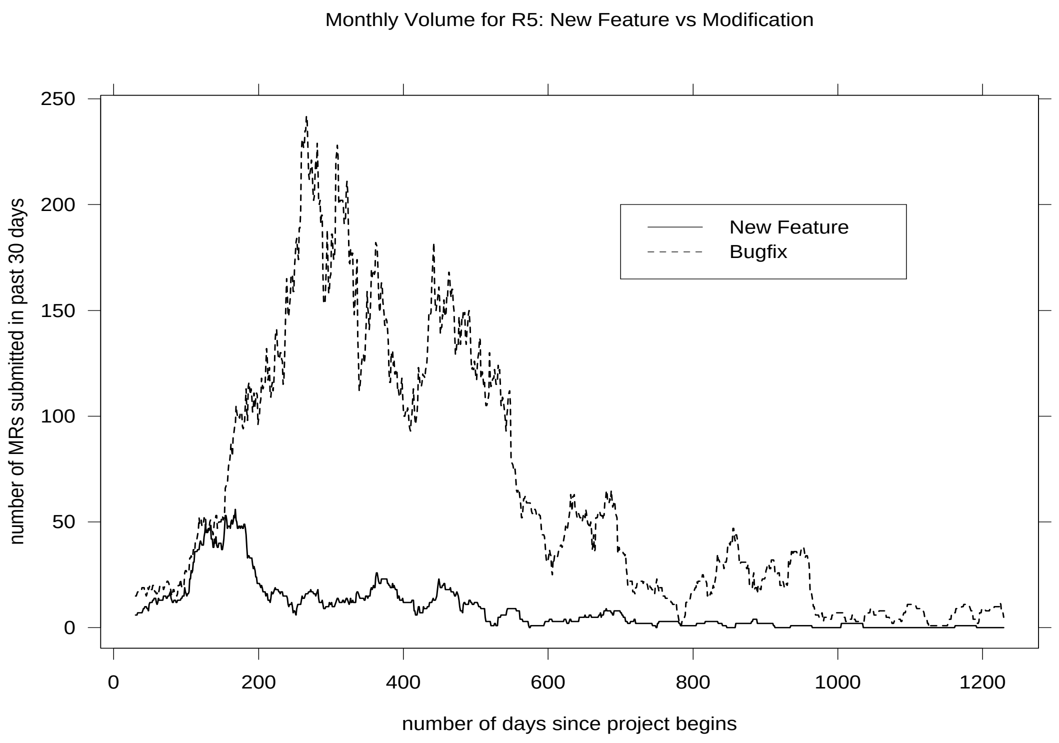

Figure 1: *The distribution of MR submit time for project R5. Each point in the graph shows the number of days since the project begins (the x-axis) and the number of MRs submitted during the past 30 days (the y-axis). The project apparently follows a waterfall model.*

The data used for this study is obtained from a large software organization within a Fortune 500 telecommunications company. For simplicity, only two types of MRs are included: new feature MRs and in-house repair MRs. Figure 1 shows how the two types of MRs are distributed over time for one of the projects we are about to study. We excluded field MRs because our goal is to model how defects are generated and we believe that in-house defects and field defects are generated by quite different mechanisms. Field MRs merit a separate study.

Before proceeding further, let us point out some of the subtleties inherent in the MR data. Strictly speaking, an MR is not a single event. Rather, it is a sequence of events that is characteristics of the software development process. An MR is first created, then assigned, submitted, tested, and finally closed. Each of these events is timestamped by either the version control system or the change management system. We decide to choose the time when the MR is submitted to the code base as the representative event associated with the MR. First, this is indeed the time when the code base is changed. All other timestamps are more likely to be subject to errors. Take the assigned time for instance, many experienced developers actually identify and fix defects before they write an MR. Thus the timestamp would not reflect when the work started. Similar argument can be made regarding the other timestamps. Throughout the rest of the paper, when we say an MR is implemented at time t, it means that the MR is submitted to the code base at time t.

3 The Poisson Model

We consider two types of MRs: Those that implement new features, and those that fix defects caused by adding new features. For a given project, let $a_k, k = 1, \ldots, K$, be the times at which new feature MRs are implemented. Typically, new features are planned in advance as business needs demand, so for our modelling purposes we assume that the timestamps $a_1, \ldots, a_K$ are deterministic. By comparison, let $t_i, i = 1, \ldots, N$, be the times at which defect fix MRs are implemented. We assume that the t_i's are random. Finally, we assume that the project starts at time 0 and the observation period is $[0, T]$. In summary, our data consists of timestamps a_k's and t_i's, where $0 < a_k, t_i < T$.

The main thesis of this paper is as follows. First, our fundamental premise is that there is a causal relationship between features and defects. Quantitative description of this relationship will not only increase our understanding of the software development process, but also enhance our ability to predict and plan future activities. However, modeling such a relationship requires non-trivial statistical inference because the relationship is not directly observable. For a given defect, we do not know which previously implemented features caused it, only that one of them did. In other words, the data we observe is at an aggregate level. The approach that we propose is to describe the data through a mixture model where each component describes the relationship between a new feature MR and its associated defects. To facilitate further illustration, we postulate the following assumptions about a software development process:

Assumption 1 *Defects discovered and fixed during development are caused by implementing new features.*

Assumption 2 *Defects caused by a given new feature are repaired at random points of time that follow a nonhomogeneous Poisson process.*

Assumption 3 *The Poisson processes generated by different features are statistically independent.*

Under these assumptions, one can describe the event data we observe as the sum of Poisson processes generated by different new feature MRs. Although mathematically the sum of independent Poisson processes is still a Poisson process, the corresponding statistical problem is quite challenging because it is technically equivalent to fitting a general mixture model. For simplicity, throughout the rest of the paper, we shall refer the model that satisfies the above assumptions as the Poisson model. In the following, we shall focus mainly on the applications of such models to software data. Those interested in general treatment of Poisson process models should consult the appropriate literature for more details (Reiss, 1993; Basawa and Prakasa Rao, 1980).

We now begin to derive the likelihood function under the Poisson model. Consider the simpliest case where one feature is implemented at time a. Let $W(t_1, t_2)$ denote the number of defect repair MRs we observe during the interval (t_1, t_2). The Poisson model implies that $W(t, t+dt)$ follows a Poisson distribution with parameter $\theta(t-a)dt$, where dt represents an infinitesimal increment in time, and $\theta(t)$ is the intensity function of the Poisson process. Accordingly, the likelihood of observing exactly one defect repair MR at time t is proportional to

$$P\{W(t, t+dt) = 1\} = \theta(t-a)dt \exp\{-\theta(t-a)dt\} \approx \theta(t-a)dt.$$

Next, suppose that K new features have been implemented at times $a_k, k = 1, \ldots, K$. Again, let $W(t_1, t_2)$ denote the number of defect repair MRs we observe during the interval (t_1, t_2). According to the independence assumption, we conclude that $W(t, t+dt)$ follows a Poisson distribution with parameter $\sum_{k=1}^{K} \theta_k(t - a_k)dt$, where a_k is the time at which the kth feature is implemented, and θ_k is the intensity function for the Poisson process generated by the kth new feature MR. This holds because the sum of independent Poisson random variables is still Poisson. Thus in the general situation, the likelihood of observing one defect repair MR at time t becomes

$$P\{W(t, t+dt) = 1\} = \sum_k \{\theta_k(t-a_k)dt\} \exp\{-\theta_k(t-a_k)dt\} \approx \sum_k \theta_k(t-a_k)dt.$$

The Poisson assumptions also imply that the likelihood of observing no defect repair MR during the interval (t_1, t_2) is equal to

$$P\{W(t_1,t_2)=0\} = \exp\left\{-\int_{t_1}^{t_2} \sum_k \theta_k(t-a_k)dt\right\}.$$

The full likelihood function, i.e., the probability of observing N defect repair MRs at times $(t_1,\ldots,t_N)$ and nowhere else during the interval $(0,T)$, is equal to

$$P\{W(0,t_1)=0\}P\{W(t_N,T)=0\}\prod_{i=2}^{N} P\{W(t_{i-1},t_i)=0\}\prod_{i=1}^{N} P\{W(t_i,t_i+dt)=1\}.$$

Combining this with the previous argument, we obtain the log-likelihood function

$$\ell(t_1,\ldots,t_N) = -\int_0^T \psi(t)dt + \sum_{i=1}^{N} \log\psi(t_i), \tag{1}$$

where $\psi(t) = \sum_k \theta_k(t-a_k)$ is the intensity function for the sum of K independent Poisson processes.

Poisson process models have been used extensively in software reliability studies (see Gokhale et al, 1996; Musa 1987, Goel 1985; Ohba 1984). However, the focus in these studies is the static finished product, assuming that it is immediately repaired as soon as the defect is discovered with no effect on the product (except elimination of the defect). The emphasis is to find as many defects as possible or decide when to stop the search for defects. By comparison, our models are fundamentally different in several respects. First, we treat the software system as a constantly changing entity instead of clearly inaccurate assumption of software as a static entity. Second, our focus is to understand the development or software change process, not the final product. Finally, by using the MR completion time rather than discovery time, our model describes the total effort that includes both the time it takes to discover a defect and the time it takes to repair a defect. Again, the repair effort is essential for project management because unlike a finished product, it is the repair activities that are taking most of the resources in a software project. It should be noted, that our model can be directly applied to address the problems of defect discovery, though such investigation is beyond the scope of this paper.

4 Parametric Inference

In the previous section, we introduced a likelihood approach for general statistical inference under the Poisson model. Specifically, the goal of inference becomes that of estimating the unknown intensity function $\theta_k(t)$ in Equation (1). For practical purposes, we can further simplify the problem by assuming that the intensity function has a parametric form.

Suppose that the total number of defects generated by a feature is finite. Then we can express the intensity function as $\theta_k(t) = \mu_k f_k(t)$, where $\mu_k > 0$ is a normalizing constant and $f_k(t)$ is a density function satisfying $\int f_k(t)dt = 1$. Intuitively, the parameter μ_k represents the total number of defects generated by the kth new feature MR, and the density function $f_k(t)$ describes how these defects will be distributed over time. Throughout the rest of the paper, we assume that $f_k(t)$ is a Weibull distribution commonly used in modelling defect discovery time, i.e.,

$$f_k(t) = \frac{\alpha}{\lambda_k}\left(\frac{t}{\lambda_k}\right)^{\alpha-1} \exp\left\{-\left(\frac{t}{\lambda_k}\right)^{\alpha}\right\}. \tag{2}$$

Intuitively, the scale parameter λ_k in (2) measures how long it takes to repair a defect generated by the kth feature. For example, when $\alpha > 1$, the Weibull distribution is unimodal with the mode given by

$$\text{mode} = \lambda_k(1-1/\alpha)^{1/\alpha}.$$

This is the amount of time it takes for defect repair activities to peak.

With the above model specifications, there will be $2K + 1$ unknown parameters remaining in the likelihood equation and the corresponding inference problem can be formidable when K is large. To further reduce the model complexity, let us assume that μ_k and λ_k are determined by some measurable attribute of the kth new feature MR, say x_k. Specifically, we assume that

$$\mu_k = \mu x_k \tag{3}$$

and

$$\lambda_k = \exp(\lambda_a + \lambda_b x_k). \tag{4}$$

This type of parametrization is commonly used in generalized regression models where the function in Equation (4) is known as a link function (see McCullagh and Nelder, 1989).

Let us now turn to the estimation of parameters $(\mu, \alpha, \lambda_a, \lambda_b)$ under the Poisson model specified by equations (1), (2), (3) and (4). If the product never changes (or $K = 1$), then the Poisson model with intensity function (2) is equivalent to the so-called extended Goel-Okumoto model in software reliability (Goel 1985). The inference problem for standard Poisson process is straightforward. When $K > 1$, our Poisson model is essentially a mixture model and the corresponding inference problem is much more challenging. To illustrate, notice that we can re-write the likelihood equation as

$$\ell = -\mu \int_0^T \sum_k x_k f_k(t - a_k)dt + N\log(\mu) + \sum_{i=1}^N \log\left\{\sum_k x_k f_k(t_i - a_k)\right\}.$$

The maximum likelihood estimate (MLE) for μ has a close form expression

$$\hat{\mu} = \frac{N}{\int_0^T \sum_k x_k f_k(t - a_k)},$$

whereas the MLE for $(\alpha, \lambda_a, \lambda_b)$ is given by

$$(\hat{\alpha}, \hat{\lambda}_a, \hat{\lambda}_b) = \arg\min \sum_{i=1}^N \log\left\{\frac{\sum_k x_k f_k(t_i - a_k)}{\int_0^T \sum_k x_k f_k(t - a_k)dt}\right\}. \tag{5}$$

What we see from Equation (5) is that the inference problem we encounter is equivalent to not just a mixture problem, but a mixture problem with truncated data. Developing a methodology that suits the above inference problem is beyond the scope of this paper. For the applications that we shall present in Section 6, we decide to use generic methods that are applicable to all parametric models. In particular, we obtain the MLE numerically by treating the likelihood function as an arbitrary function (we can use the closed form expression for $\hat{\mu}$ given above to reduce the number of parameters in the maximization problem).

In theory, the standard error of the MLE can be estimated by calculating the inverse of the Fisher matrix. However, this approach seems less practical when the likelihood function is complex and the calculation of second order derivatives becomes demanding. Thus for convenience, we use the delete-one jackknife method to get an approximation of the standard error. To be specific, suppose that θ is a generic parameter and $\hat{\theta}$ is an estimate. Let $\hat{\theta}_{(-j)}$ denote the MLE of θ using all but the jth data point (in our case it represents the jth repair MR). The standard error of $\hat{\theta}$ can be estimated by the square root of

$$(N-1)J^{-1} \sum_{j=1}^J \{\hat{\theta}_{(-j)} - \hat{\theta}\}^2,$$

where the index j is a random number from 1 to N selected without replacement.

Finally, we can also derive a goodness-of-fit test to see if the model fits the data. To this end, suppose that we partition the observation interval $(0, T)$ into mutually disjoint sub-intervals $A_j, j = 1, \ldots, J$. Let Y_j

denote the number of defect repair MRs submitted within the interval A_j. According to the Poisson model, the random variables $Y_j, j = 1, \dots, J$ will be mutually independent following Poisson distributions. Define

$$Z_j = (Y_j - \eta_j)/\sqrt{\eta_j}, \tag{6}$$

where $\eta_j = E(Y_j) = \mu \int_{A_j} \sum_k f_k(t)$. Under reasonable conditions (e.g., the intervals being large enough), the transformed variables $Z_j, j = 1, \dots, J$ are approximately iid following standard normal distributions. The Poisson assumption can therefore be tested through a QQ-plot.

5 Prediction

Frequently, a software development organization has to assess the status of an ongoing project and find out, among other things, whether the project can be finished on schedule. There is a great deal of uncertainty in such assessment due to the lack of objectiveness and transparency. Often, the determining factor is expert opinion rather than empirical evidence. One of the advantages of modeling the software process at the MR level is that data from historical projects can be analyzed to shed light on the current project. In particular, we can use the model to predict how many defect repair MRs will have to be fixed during a future time interval.

To see how this can be accomplished, let t denote the current time. Let N_t be the number of defect repair MRs that will be generated after t. Let $a_k, k = 1, \dots, K$ denote the time points at which the new feature MRs will be implemented. Then under the Poisson model, the random variable N_t follows a Poisson distribution with

$$E(N_t) = \mu \sum_{k=1}^{K} \int_t^\infty f_k(s - a_k) ds. \tag{7}$$

Under the parametric model of Section 4, the above equation becomes

$$E(N_t) = \mu \sum_{k=1}^{K} \exp\left\{ -\left(\frac{t - a_k}{\lambda_k} \right)^\alpha \right\}.$$

Suppose that there is a target deadline for the project, say at time T. Then one way to assess the project status at time t is to estimate the number of defect repair MRs that will need to be fixed before the deadline. This is given by $E(N_t) - E(N_T)$. In addition, we can also look at $E(N_T)$, which gives us a sense of what the product quality will be if we end the project at time T.

In practice, a number of decisions have to be made in order for this approach to be viable. First, as in all prediction problems, we need to assume that past is indicative of the future. For a mature software organization, this might be true if the culture, the process, and the product remain relative stable over time. On the other hand, the environment for software development could also change rapidly due to technology innovation or shift in market conditions. In any event, making prediction is a non-trivial task that requires in depth understanding of the past and careful evaluation of future possibilities. Analyzing historical data is only the beginning. Next, past projects vary in size, duration, and perhaps many other aspects. For any past project, we can estimate the model parameters according to the procedure described in Section 4. These estimates can then be plugged into Equation (7) to predict the outcome of the current project. But which past project should we use? In principle, it is possible to combine data from different projects, or even combine historical data with partial data from the current project (i.e., an adaptive approach) using Bayesian techniques or through a hierarchical model. But such combination could be performed in numerous ways and a thorough treatment is ouside the scope of this paper.

In the next section, we apply the above prediction approach to a project that appears to be different from most historical projects and show that the basic approach, when combined with project specific knowledge, seems to work well.

6 Application

In this section, we apply the Poisson model and the inference methods developed in previous sections to real data obtained from a large software development organization within a Fortune 500 telecommunications company. The data consists of change logs and MRs (see Section 2) for six projects that vary in size and duration. For confidentiality, we label the projects arbitrarily as R1 to R6. The total effort includes more than 200 developers working on approximately 13000 MRs during a period of more than 5 years. For each MR, we use the timestamp when the MR is submitted to the code base in our analysis. A high level summary of this data is given in Table 1. One can see, for example, that on average each new feature MR generates more than 5 defect repair MRs. How long does it take to discover and to repair these defects? Figure 1 depicts the temporal distribution of new feature and defect repair MRs for project R5. An obvious observation is that there is a delay effect at the aggregate level. Defects are not repaired immediately after the feature is implemented. In fact, there is almost a 3-month gap between the peaks of new feature and defect repair activities.

Table 1: *The first two columns show the number of new feature MRs and the number of defect repair MRs for each project. N_{dev} is the number of developers working on the project. KNSCL is the number of source code lines changed (in 1000's).*

Project	K	N	N_{dev}	KNSCL
R1	90	534	91	206
R2	117	641	105	182
R3	460	2097	164	2336
R4	432	1830	151	1384
R5	410	2805	162	1194
R6	324	1205	145	875

6.1 The Basic Model

Let us begin with a simple case where we ignore the possible variation between different new feature MRs and treat them as identical events. This amounts to setting $x_k = 1$ in Equations (3) and (4), or $\mu_k = \mu$ and $\lambda_k = \lambda$ for $k = 1, \ldots, K$. We shall refer to this reduced Poisson model as the basic model.

Table 2: *Parameter Estimates from the Basic Model. The numbers in the parentheses are the standarded errors estimated by the jackknife method. The unit of $\hat{\lambda}$ is number of months.*

Project	$\hat{\mu}$	$\hat{\alpha}$	$\hat{\lambda}$	ℓ
R1	5.8667 (0.0257)	1.6221 (0.0863)	3.2934 (0.1267)	1522.72
R2	5.4939 (0.0016)	2.2111 (0.1968)	3.0160 (0.1072)	1898.13
R3	4.6023 (0.0093)	0.4742 (0.0164)	1.6786 (0.1298)	6691.68
R4	4.2905 (0.0141)	0.4858 (0.0316)	1.5107 (0.1540)	6255.35
R5	6.7481 (0.0014)	1.5056 (0.0907)	4.7106 (0.1528)	9158.29
R6	4.6368 (0.2524)	0.3630 (0.0194)	7.0513 (2.1622)	3514.07

Table 2 shows the result of fitting the basic model to data from the six projects. Our first observation is that the parameter μ, which measures the defect rate per feature MR, is relatively constant regardless of the project size. Thus larger projects are not necessarily more buggy. By contrast, the other model parameters show substantial variations across projects. An interesting but less obvious observation is that the scale parameter λ appears to be highly correlated with the defect rate μ. Specifically, a linear relationship holds so that $\lambda = 3.26 + 0.75\mu$ with $r = 0.99$ if we exclude R6. What this means is that if the per feature defect rate increases by 1, the defect repair interval will increase by 3 weeks. We need to point out that this is a 3-week delay for each new feature. The aggregate impact to the project will be much larger.

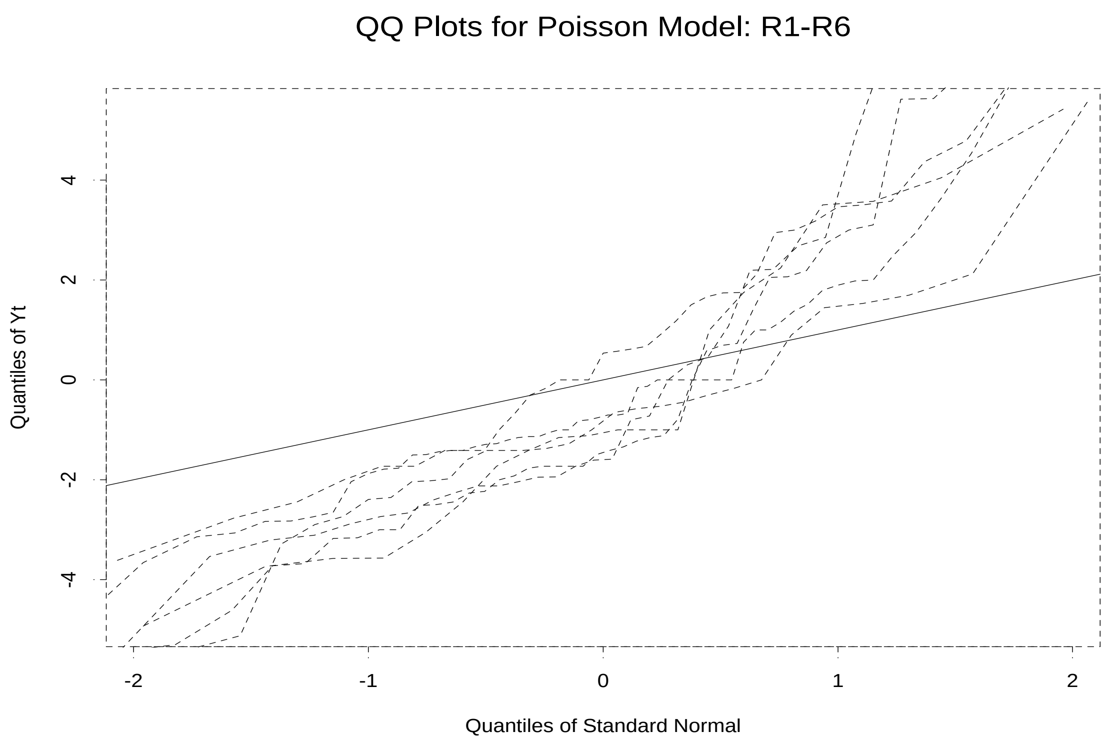

Figure 2: *QQ-plot for goodness-of-fit test. The dashed lines correspond to the results for fitting the basic models to the six projects. The solid line shows what the plot should look like if the model fits well. The time interval used in Equation (6) is 30 days.*

Figure 2 shows the QQ-plot for fitting the basic model to all six projects using the method described in Section 4. Ideally, if the model fits well, one should see a straight line that crosses the origin with a slope of 45 degrees. There seems to be a small negative bias for all but one models, indicating that the volumes estimated by the model will be slightly lower than actual observations. In addition, the slopes are clearly larger than 45 degrees, which implies that the actual variation in the data is higher than what the Poisson model allows. The interpretation of such over-dispersion might lead to interesting hypotheses that warrant further investigation.

6.2 The Effect of MR Difficulty

The next example is aimed at testing the hypothesis that implementing difficult new features will generate either more defects or more difficult ones. Here we use the effort, measured in person-months, as an indirect measure of difficulty. Although effort at the individual MR level is usually not observable, it can be estimated from high level data (see Graves and Mockus, 2001). In the following, we shall fit a Poisson model with x_k being the estimated effort spent on the kth new feature MR. For convenience, we refer to such a model as the effort model.

Table 3 shows the result of fitting the effort model to the same set of project data. First, we notice that the parameter estimates for μ is highly significant. Thus difficult new features indeed generate more defects, about 13 more defects for each extra person-month spent on new features. Second, the shape parameter α is larger than 1 for all but one projects, suggesting that there is a certain delay between the implementation of new feature and the subsequent defect repair activities. Finally, the estimate for λ_b is either positive or statistically indistinguishable from zero. In other words, the model shows that difficult features tend to generate defects that take longer time to repair.

Table 3: *Parameter Estimates from the Effort Model. The numbers in the parentheses are the standarded errors estimated by the jackknife method.*

Project	$\hat{\mu}$	$\hat{\alpha}$	$\hat{\lambda}_a$	$\hat{\lambda}_b$	ℓ
R1	13.7840 (0.0848)	2.0670 (4.3882)	2.2465 (12.3711)	-1.5515 (19.5232)	1507.68
R2	15.1212 (0.0016)	2.1213 (0.1605)	1.2551 (0.0479)	0.1039 (0.0933)	1926.56
R3	13.6012 (0.0394)	0.5800 (0.0335)	-2.352 (1.7433)	5.4534 (2.6585)	6672.29
R4	11.6102 (0.0011)	1.2048 (0.0964)	0.8518 (0.1872)	1.0597 (0.3012)	6234.35
R5	17.3778 (0.0998)	1.7743 (0.1023)	0.1302 (0.1611)	2.4879 (0.2499)	10101.21
R6	9.8135 (0.0925)	1.0877 (0.1816)	-0.2215 (0.3068)	3.0052 (0.5131)	3526.93

6.3 Prediction Based on the Basic Model

Figure 3 shows the data from a new project that the same development organization has undertaken recently. The cyclical pattern is evidence that a spiral development model was used for the project. Features appear to have been divided into groups and implemented in phases. However, the aggregate effect appears to be the same as before, i.e., defect repair activities follow roughly the same pattern as feature implementation, except that it is delayed by several months and the scale is an order of magnitude larger. The target release date is $t = 550$ in Figure 3. An assessment was made at time $t = 350$ to determine whether the project will be late. The six projects mentioned at the beginning of this Section had already been finished by the time the new project started. The question arises whether we can use historical data to shed light on the status of the current project.

In retrospect, one realizes that the new project is more difficult than most of the previous projects because it is introducing a major new product, rather than a new release of an existing product. Suppose that one had such information prior to the beginning of the project, can we produce better prediction based on this prior knowledge? The prediction method described in Section 5 is straightforward to use. All it takes is to substitute the parameters in (7) by some empirical estimates. But which past project should we use to derive the estimates? Common sense suggests that one should look for a project that is equivalent in size to the current project. According to Table 2, project R5 seems to be a good candidate because it has the highest defect rate per feature MR and the longest repair interval. Under the basic model, the estimated values of (μ, α, λ) for project R5 are $(6.7481, 1.5056, 4.7106)$. Figure 4 shows that the predicted weekly volume using these parameters is far below the actual observation. As an experiment, we multiplied the above parameters by a factor of 2. This amounts to assuming that the new project is twice as difficult as R5. The resulting prediction, as shown in Figure 4, turns out to be quite accurate.

7 Discussion and Future Work

Software systems are entities that change constantly throughout their life cycle. We focused on understanding the relationship between different types of changes and the effect of these changes on project status. The premise that defects discovered and fixed during development are caused by implementing new features allowed us to decompose a software development project into many sub-cycles, one for each new feature

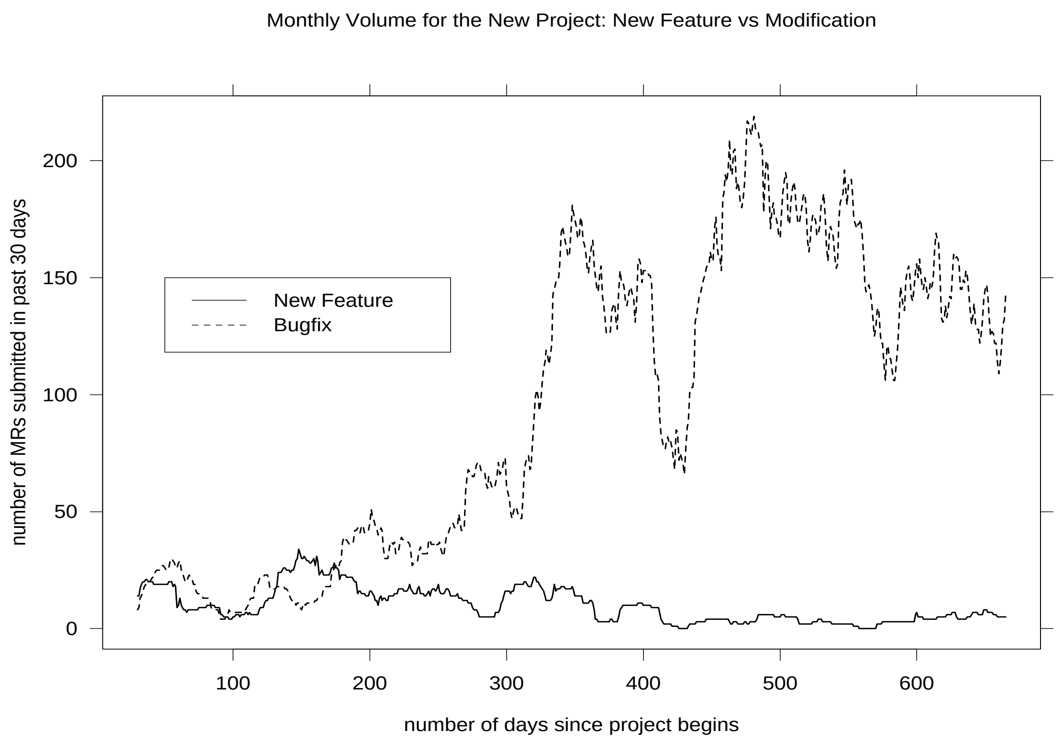

Figure 3: *The distribution of MR submit time for the new project. This is a more difficult project than the one shown in Figure 1. Also, it appears that the project is divided into phases and development follows a spiral model.*

MR. To solve the corresponding statistical problem we proposed a Poisson process model to describe the relationship between new features and the associated defects. We showed that statistical inference under the proposed model is equivalent to the inference problem under a general mixture model with truncated data. We applied the model and the corresponding inference methodology to several projects and the main results are summarized as follows:

Based on the goodness-of-fit test (Figure 2), we find that the model fits the data quite well regardless of project size and duration. There is an obvious over-dispersion effect that merits further study. But this does not affect the unbiasedness of volume estimates derived from the model. The intensity function of the Poisson model provides a good description of how defect repair MRs are distributed over time after a new feature MR is implemented. The main characteristics of the intensity function, as represented by the parameters in a Weibull density function, remain unchanged across projects. For instance, the fact that $\alpha > 1$ for nearly all projects indicates that there is always a certain amount of delay between feature implementation and defect repair activities. The details, however, can vary for different projects. For example, we find that the time it takes to repair a defect has very high correlation with the defect rate (see Table 2). The same is also true at the feature level, i.e., we find that defect repair MRs generated by more difficult new features take longer time to repair (see Table 3). When using the model for prediction, we find that some initial estimate of project size plays a critical role.

We conclude this paper by pointing out a number of open issues that need to be resolved in future studies. Due to space limitation, we will not pursue these topics in any details.

From a methodology point of view, there is potential limitation on how far we can generalize the proposed modeling approach. For example, suppose we want to extend the model by allowing Equations (3)

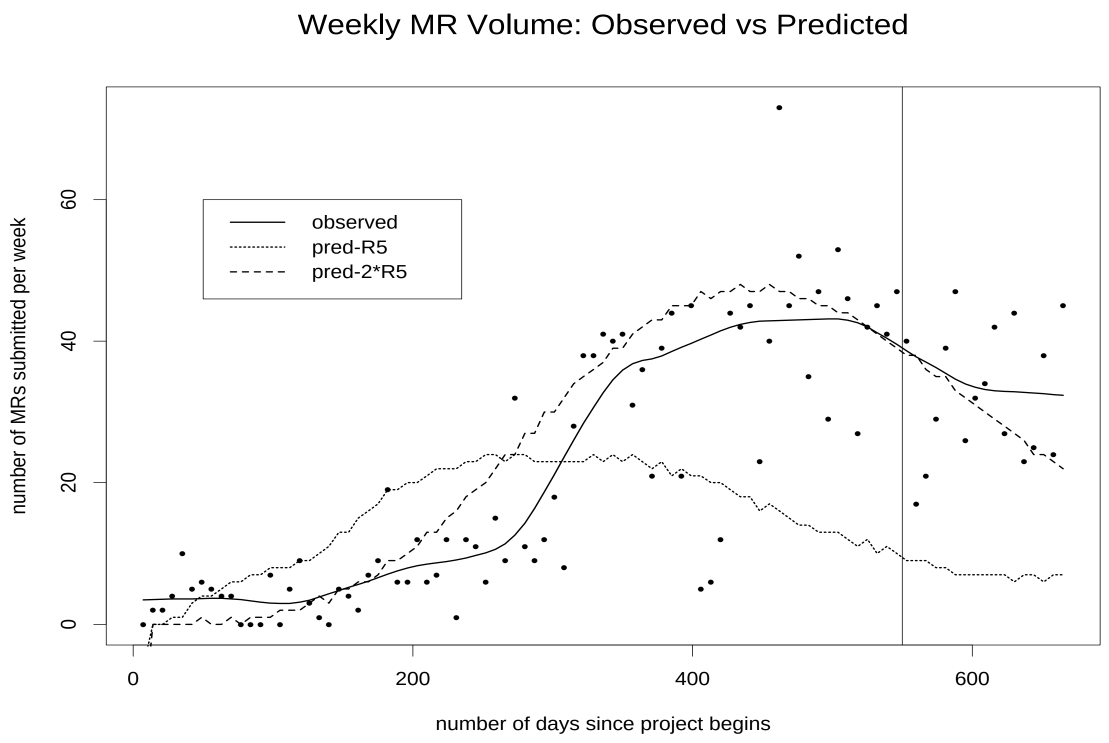

Figure 4: *The solid curve shows the observed number of defect repair MRs submitted per week smoothed using the lowess method. The dotted lines shows the prediction using project R5 parameters. The dashed line shows the prediction based on 1.5 times the parameters of project R5. The vertical line marks the target release date.*

and (4) to be a general linear functions with multiple covariates. Conceptually, the modeling approach is still valid. But computationally, it may not be viable to use a generic algorithm on arbitrary inference problems. More research is needed to find out how to expand the scope of the inference methodology.

From a practical point of view, one can argue that prediction is the ultimate test of any proposed methodology. Ideally, a prediction method in software development should be adaptive to new data. If possible, the partial history of an ongoing project should be combined with historical data for better prediction, especially when data is collected continuously as is the case in most software development projects. Unfortunately, the proposed modeling approach does not work well for unfinished projects. According to Equation (5), the likelihood function is truncated at T. This means that inference results based on unfinished projects (i.e., small T) will be highly unreliable. Methods that deal with this particular type of censored data will be an important step towards reducing the subjectiveness of model based predictions.

Finally, our results suggest that a Poisson model with over-dispersion might be a better description of software project data (see Figure 2). This leads to some interesting hypotheses that warrant further investigation. For instance, it is well known that a marked Poisson process will generate over-dispersed Poisson data (McCullagh and Nelder, 1989). To be specific, suppose that it is not the number of MRs that matters, but some other hidden quantity. Thus we observe data of the form $\sum_{j=1}^{N} Z_j$, where the Z_j's are positive iid random variables and N follows a Poisson distribution. For example, we can let Z_j be the effort spent on the jth MR. For a software development organization, predicting the amount of effort will be much more useful than predicting the number of MRs.

8 Conclusions

A number of important lessons can be drawn from this investigation. First, modeling the relationship between new feature and repair MRs leads to an approach that unifies the reliability and estimation models in software engineering.

Second, in contrast to existing models, our model is based on data that is ubiqutous in software projects implying that it can be applied for almost any software project without the costly and often prohibitive option of collecting additional data.

Third, our results show that the overall size of a project appears to be important in many aspects of modeling and prediction (the fundamental assumption of the COCOMO model). However, unlike the COCOMO model, we do not require an absolute measure of size as input data. Instead, what we need is a rough estimate of relative size. The example that we studied suggests that a reasonably accurate prediction can be derived provided that someone with knowledge of the project can say that "*the new project is expected to be x-percent larger or more difficult than R5*".

Fourth, the above investigation also shows that methodology alone is not enough to solve a prediction problem. Some sort of subjective knowledge, including project specific information, has to be incorporated into the prediction method. The value of modeling the MR data, and what is encouraging about our prediction method, is that one can reduce the amont of subjectiveness to a great extend.

Finally, even in the absence of subjective size estimates, one can still use our prediction method to produce a range of "what-if" scenarios that will be helpful to decision makers.

We expect such models to be of value to understand the complexities of software projects. The simplicity of the model and the wide availability of the data should help deploy such models as tools for managing and understanding large software projects.

References

[1] Basawa, I.V. and Prakasa Rao, B.L.S. (1980), *Statistical Inference for Stochastic Processes*, New York: Academic Press.

[2] Beck, K. (2000), *Extreme Programming Explained: Embrace Change*, Boston, MA: Addison-Wesley.

[3] Boehm, B. (1981), *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall.

[4] Brooks, F.P. Jr. (1995), *The Mythical Man-Month: Essays on Software Engineering* (Anniversary Edition), Boston, MA: Addison-Wesley.

[5] Dalal, S.R. and Mallows, C.L. (1988), "When should one stop testing software?", *Journal of American Statistical Association*, 83:872-879.

[6] Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., and Mockus, A. (2001), "Does Code Decay? Assessing the Evidence from Change Management Data", *IEEE Transactions on Software Engineering*, Vol.27, No.1., pp 1-12.

[7] Goel, A.L. (1985), "Software Reliability Models: Assumptions, Limitations, and Applicability", *IEEE Transactions on Software Engineering*, Vol.11, No.12.

[8] Gokhale, S.S., Marinos, P.N. and Trivedi, K.S. (1996), "Important Milestones in Software Reliability Modeling", *Proc. Software Engineering and Knowledge Engineering (SEKE) '96*, Lake Tahoe, NV.

[9] Graves, T.L. and Mockus, A. (2001), "Identifying productivity drivers by modeling work units using partial data", *Technometrics*, (43)2: 168-179.

[10] Kingman, J.F.C. (1993), *Poisson Processes*, Clarendon: Oxford.

[11] McCullagh, P. and Nelder FRS, J.A. (2nd eds.) (1989), *Generalized Linear Models*, New York: Chapman and Hall.

[12] Musa, J. D., Iannino, A., and Okumoto, K. (1987), *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill.

[13] Ohba, M. (1984), "Software Reliability Analysis Models", *IBM J. Res. Develop.*, Vol.28, No.4.

[14] Reiss, R.D. (1993), *A Course on Point Processes*, New York: Springer-Verlag.

[15] Rochkind, M.J. (1975), "The Source Code Control System", *IEEE Transactions on Software Engineering*, Vol.1, No.4, pp. 365-370.