Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems

Paul Luo Li, Mary Shaw, Jim Herbsleb
Institute for Software Research International
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{paul.li,mary.shaw,jim.herbsleb}@cs.cmu.edu

Bonnie Ray, P.Santhanam
Center for Software Engineering
IBM T.J. Watson Research Center
Hawthorne, NY 10532
{bonnier,pasanth}@us.ibm.com

ABSTRACT

Defect-occurrence projection is necessary for the development of methods to mitigate the risks of software defect occurrences. In this paper, we examine user-reported software defect-occurrence patterns across twenty-two releases of four widely-deployed, business-critical, production, software systems: a commercial operating system, a commercial middleware system, an open source operating system (OpenBSD), and an open source middleware system (Tomcat). We evaluate the suitability of common defect-occurrence models by first assessing the match between characteristics of widely-deployed production software systems and model structures. We then evaluate how well the models fit real world data. We find that the Weibull model is flexible enough to capture defect-occurrence behavior across a wide range of systems. It provides the best model fit in 16 out of the 22 releases. We then evaluate the ability of the moving averages and the exponential smoothing methods to extrapolate Weibull model parameters using fitted model parameters from historical releases. Our results show that in 50% of our forecasting experiments, these two naïve parameter-extrapolation methods produce projections that are worse than the projection from using the same model parameters as the most recent release. These findings establish the need for further research on parameter-extrapolation methods that take into account variations in characteristics of widely-deployed, production, software systems across multiple releases.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Empirical Studies, Metrics, Reliability Engineering

Keywords

Defect modeling, empirical research, COTS, open source software, maintenance resource planning, software insurance

1. INTRODUCTION

Defect occurrences not only create problems for software consumers, but also cause problems in maintenance planning for software producers. The costly consequences of defect occurrences have increased interest in insuring software consumers against the associated risks.

SIGSOFT'04/FSE-12, Oct. 31–Nov. 6, 2004,Newport Beach, CA, USA.

Defect-occurrence projection is crucial to the development of methods for managing the risks associated with defect occurrences. Accurate defect-occurrence projections can help software maintenance planners to better allocate resources and will be a major step towards novel risk-mitigation techniques for software consumers, such as software insurance.

We examine software systems that businesses are increasingly dependent upon. These systems are multi-release, multi-platform, and widely-deployed, such as COTS and open source software. It is generally accepted that these widely-deployed, production, software systems (WPSSs) are not defect free and that there is a need to manage the risks associated with the defect occurrences.

We empirically address two questions that are important for defect-occurrence projection:

- Is there a type of defect occurrence model that provides a good fit to defect-occurrence patterns across multiple releases and in many organizations?
- Given such a model, how can model parameters for a new release be extrapolated using historical information?

Our findings provide a basis for a defect-occurrence projection method for WPSSs that is robust across many organizations and development styles.

We use data from a diverse sample of WPSSs including two different types of software systems (middleware and operating systems) developed with two different development styles (commercial and open source). We gather data from twenty-two releases: eight releases of a commercial operating system, three releases of a commercial middleware system, eight releases of an open source operating system (OpenBSD), and three releases of an open source middleware system (Tomcat).

We examine characteristics of WPSSs that can change between releases and that may cause variations in defect-occurrence patterns. The characteristics we consider are release content, development process, adoption and usage patterns, and software and hardware configurations in use. These considerations are not modeled well in prior research in this field.

We examine how parameterizations of a set of candidate defect-occurrence models taken from the literature account for possible variations in defect-occurrence patterns across multiple releases and how two commonly-used naïve parameter-extrapolation methods account for the variations. We hypothesize:

- The Weibull model is better than other candidate models at modeling defect-occurrence patterns for multiple releases of WPSSs.
- Naïve parameter-extrapolation methods, moving averages and exponential smoothing, extrapolate model parameters

that produce inadequate defect-occurrence projections for new releases of WPSSs.

Determining the preferred defect model is important for defect-occurrence projection because it may allow us to understand and quantify the effects of changes in characteristics of WPSSs across multiple organizations and development styles. Verifying that naïve parameter-extrapolation methods are inadequate is important because it will motivate additional research questions on new methods for parameter extrapolation. Our results show that the Weibull model is the preferred model and that naïve parameter-extrapolation methods are inadequate.

We begin by providing background and descriptions in Section 2. We present analyses that support our two hypotheses and the empirical results in Sections 3 and 4. We conclude with validity issues and future work in Section 5.

2. BACKGROUND AND DESCRIPTIONS

We are interested in real-world software systems deployed today that are of key business interest to users, such that there are users who pay for maintenance contracts and who may be willing to pay to insure against defect occurrences.

2.1 Defect occurrence

We define a *defect occurrence* as a user-reported problem that requires developer intervention to correct. This is the observable event of interest for both maintenance and insurance purposes.

The operational definition of a defect occurrence varies across organizations. In this paper, we use the same approach to analyze defect occurrences in different organizations and show that our approach is resilient to organizational differences. The commercial software development organizations measure faults and failures, described in Section 2.2.1, while open source software projects track user-submitted bug-reports, described in Section 2.2.2. Our findings support the idea that a common defect-occurrence projection method for WPSSs can be used across many organizations and development styles.

We are interested the *defect-occurrence pattern*, which is the rate of defect occurrence as a function of time over the lifetime of a release. We define the *lifetime* of a release as the duration of time between when a release becomes generally available and when there are no defect occurrences reported to the software development organization for three consecutive time intervals. Determining the lifetime of a release is discussed in detail in [15, Appendix C].

Our focus on the defect occurrence pattern is different from previous research on the total number of defects [5] and the normalized defect-occurrence rate. The normalized defect-occurrence rate is the rate of defect occurrences normalized with respect to the number of deployed systems and the usage amount over the lifetime of a release [4][18][19][20][29]. Knowing only the total number of defects is inadequate because resource allocation for maintenance planning and cash reserve management for insurance both require knowledge of how many defects are going to occur in a given time period. The normalized defect-occurrence rate is unsuitable because it requires accurate measurements of deployment and usage patterns. As we explain in the next section, unknown deployment and usage patterns are properties of WPSSs.

2.2 Widely-deployed production software systems

We are interested in *widely-deployed, production, software systems* (WPSSs), which are software systems with the following properties:

- The software system is used in many software and hardware configurations (some unforeseen).
- The deployment and usage patterns of the software system are unknown.
- The development process of software system has constraints (such as scheduling and resource constraints).
- The contents of the software system change over time.
- The software system has multiple releases.

2.2.1 Commercial software systems

Nearly all COTS software systems have the properties of WPSSs. COTS software systems are not developed with one client in mind but rather to be sold on the open market and to be used by many clients [2]. The systems are typically built to run on multiple hardware platforms, to be compatible with many different hardware devices, and to be compatible with many other software systems. Constraints exist on the development process, such as pre-set release dates and limited resources. The software development organization has limited information on who is going to purchase the software system and puts out successive versions of the software system, which implements new functionality and improvements [12].

The two COTS software systems we examine are developed by two different divisions of IBM. The operating system is a mature product with many years of presence in the marketplace. The commercial middleware system has a few years of deployment history and a growing customer base.

The defect-occurrence data collected are code-related problems discovered and reported by customers after deployment. A more detailed description of the data collection process is in [15, Appendix E]. The defect-occurrence data for the operating system contain unique field defects that led to code changes by the product support organization. The defect-occurrence data for the middleware system contain all field defects (which may not be unique) that lead to code changes. The defect-occurrence data are processed and aggregated, so for each data set we use the interval in that data set. The time interval for the middleware system is a month and the time interval for the operating system is a quarter.

2.2.2 Open source software systems

Open source software systems have all the properties of WPSSs. A consortium of user-developers with different needs (generally with one person or a core group leading development) develops an open source software system. A successful project has diverse user-developers that develop and test the software system, port the software system to many platforms, make it operable with many devices, and make it compatible with other software systems. Hence, the number of active user-developers constrains an open source project. The software system can be downloaded and used anonymously, so there is limited knowledge about the users. Open source software systems usually evolve in successive releases to satisfy the needs of its user-developers [25].

The open source software systems we examine are developed by successful open source projects with many user-

developers from around the world. The open source operating system is OpenBSD. OpenBSD is a Unix-like operating system that emphasizes portability, standardization, correctness, proactive security, and integrated cryptography [21]. The open source middleware system is Tomcat, which is one of the products developed by the Jakarta Project. Tomcat is the servlet container used in the official reference implementation for the Java Servlet and JavaServer Pages technologies [7].

The defect-occurrence data collected are bug reports submitted by users via the web based bug-tracking system. A more detailed description of the data collection process is in [15, Appendix E]. Unlike users of a commercial software system, user-developers of an open source software system can obtain the software system anytime during development and submit bug reports. Many bugs are duplicates, user mistakes, or otherwise invalid submissions that do not require code changes. However, we include all user-reported bug reports in our data set because a member of the core development team examines each bug submission and decides upon a course of action. The time interval used for both open source software systems is a month.

2.2.3 Related work

Much of the prior research in software reliability has been conducted on systems where the testing environment and the deployment environment are similar. Similarities like software and hardware configurations and usage patterns have allowed researchers to extend defect-occurrence patterns from development into the field. Lyu in [16] provides a comprehensive review of previous works. Lyu explains model origins, states modeling assumptions, and classifies commonly used reliability models. We borrow mathematical models and statistical tools from prior research, however extending defect-occurrence patterns is inappropriate for WPSSs given properties discussed in Section 2.2.

Current efforts to certify software are very similar to prior research in software reliability engineering. Voas advocates certifying commercial software for use in a customer's environment [28], and Wallnau et. al. at the Software Engineering Institute are conducting research on predictable assembly from certified components [14]. Both approaches test software in the customer's environment then extend results into usage. These approaches account for several environmental variables and make statistical guarantees about various properties. However, we feel that cost, variance, and unforeseen confounds at the single-customer level might make such methods impractical. We do however, leverage relationships between characteristics of software systems and defect occurrences discovered in their research.

Several on-going research efforts examine defect occurrences across multiple releases. Ostrand and Weyuker use software content and development process measures to predict faulty files in multiple releases of two software systems at AT&T [22]. Their model predicts the top 20% faultiest files, which they show to capture around 80% of faults found during development and in the field. The COQUALMO project at USC uses COCOMO II data to estimate the total number of defects in a software system [5]. Their model uses size metrics and various process modifiers. The process modifiers measure aspects of the defect injection process and the defect removal process. Jones et. al at Nortel use the percentage of deployed systems with a module installed as a surrogate for usage along with software content measures to predict the likelihood that a module will be faulty in the field [10]. They correctly identify approximately 70% of the faulty modules while misidentifying less than 26% of the faulty modules. These research projects contribute knowledge about the important predictors of defect occurrences. However, none of the projects examines the defect-occurrence pattern. As noted in Section 2.1, risk mitigation techniques need to know both the expected total number of defect occurrences as well as when those defect occurrences happen in the lifetime of a release.

Mockus et. al. use software content and development process information captured in the change management and CVS systems to predict the amount of repair effort and the delay until the effort is needed for eleven releases of a telecommunications software system at Avaya [17]. They assume that field repair effort is proportional to development effort and estimate a delay factor and an effort multiplier. Our work is similar. However, our focus is on defect-occurrence patterns, while the focus at Avaya is on effort.

Popstajanova et. al in [23] uses architecture, utilization, and control flow information to predict the likelihood of defect occurrences. Unlike their white box approach, our approach is black box.

Previous works have also compared defect-occurrence models. Jones compares various models and the MLE and least squares model fitting methods for ten releases of a telecommunications software system at Northern Telecom and Bell Canada [9]. He concludes that the Logarithmic model, fitted using the least squares method, produces the best results and that the least squares method is superior to the MLE method overall. Jones considers a Weibull process model. Wood compares eight mathematical models fitted using the least squares method for four releases of a software system at Tandem computers [29]. Wood considers the Weibull model but finds that the Exponential is the best model in his environment. Their works lack an underlying theory explaining why a model is superior in their environment, and they do not replicate their experiment at other organizations. We develop a theory for the causes of variation in defect-occurrence patterns for WPSSs and compare models across multiple releases, multiple organizations, and multiple products.

There appears to be no published work on projecting defect-occurrence patterns for open source projects. We note that software systems are becoming so complex and expensive that few organizations have the resources to build custom systems. More and more organizations reply on COTS or open source software systems, which is the focus of this paper.

2.3 Characteristics of WPSSs

Content, development process, deployment and usage patterns, and software and hardware configurations in use are characteristics of WPSSs that tend to change, sometimes dramatically, between releases. The effect of changes on defect-occurrence patterns will dictate which defect-occurrence model is best suited to model defect-occurrence patterns in multiple releases and which parameter-extrapolation methods are effective. We use results from previous work in software reliability engineering and intuitive arguments to provide evidence that the characteristics listed above can influence defect-occurrence patterns across multiple releases.

2.3.1 Content

Software content affects defect-occurrence patterns across multiple releases [11][17]. Successive releases incrementally add features and implement internal changes, such as refactoring. Some modifications may be more difficult and defect prone than others, which may cause more defect occurrences over a longer time period. Depending on the similarity of content changes, the defect-occurrence pattern of a release may be similar to one release but be substantially different from another.

2.3.2 Development process

The development process affects defect-occurrence patterns across multiple releases [12][7]. Although the development process of most organizations changes slowly over time, when combined with other factors the development process may have significant impact on defect-occurrence patterns. Insufficient and varying testing resources and schedule pressure may lead to ineffective defect removal. This can cause defects to linger in the system and can cause blocking, which occurs when one defect masks the presence of other defects. However, problems with the development process may not be present to the same degree in every release, which may cause defect-occurrence patterns to vary.

2.3.3 Deployment and usage patterns

Deployment and usage patterns affect defect-occurrence patterns across multiple releases [3][16]. The total number of deployed systems and the pattern of deployment dictate how much usage and how many usage patterns are possible. Deployment may be different for each release because users cannot be forced to adopt the latest release. Some users may adopt every release, while others may only adopt releases that contain important functionality. In addition, some users may adopt a release immediately, while other may delay adoption. The usage patterns may also be different from release to release and from user to user. Some users may not heavily exercise the software until the software has shown to be satisfactory under normal operating conditions. Deployment and usage patterns dictate how heavily the system is exercised, which may cause variations in defect-occurrence patterns.

2.3.4 Software and hardware configurations in use

Software and hardware configurations in use affect defect-occurrence patterns across multiple releases [14][28]. The software development organization has limited knowledge about users, and thus may not be able to test all possible hardware and software configurations. Furthermore, comprehensive testing may not be feasible given economic and scheduling constraints. Therefore, the software may have defects that are specific to certain configurations, software interactions, and other special conditions, such as malicious attacks.

3. DEFECT-OCCURRENCE MODELS

A single type of defect-occurrence model may be able to model defect-occurrence patterns across multiple releases of a wide variety of WPSSs. We believe such a model exists despite possible changes in characteristics discussed in Section 2.3 because of similarities between releases and properties common to all WPSSs. Bassin and Santhanam at IBM have shown that successive releases have similarities in the functionalities implemented, the development organization, and users with similar usage patterns [1]. All WPSSs also share common properties mentioned in Section 2.2.

In order for a model to be widely applicable, it needs to have parameterization that can account for variations in defect-occurrence patterns. In this section, we develop and use real-world data to empirically test the hypothesis that the Weibull model is better than other candidate models in modeling defect-occurrence patterns for multiple releases of WPSSs.

3.1 Candidate models

We are interested in the defect-occurrence pattern. Therefore, we are interested in models that model the number of defect occurrences in each time period over the lifetime of a release. We consider the Exponential model, the Gamma model, the Logarithmic model, the Power model, and the Weibull model. These models are promising because prior research in software reliability engineering has shown each model to be effective at modeling defect-occurrence patterns at a software development organization [9][16][29]. Each model is parametric. The number of defect occurrences during the t-th time interval is determined by the model parameterization and the current time interval. The number of defect occurrences within a time interval is modeled as a non-homogenous Poisson process with a stationary defect rate $\lambda(t)$. Table 1 lists the models. Lyu [16] provides details about the models, including researchers who have developed and applied the models in practice.

3.2 The Weibull model

The Weibull model can account for different increasing and decreasing trends, which reflect initial increases and eventual decreases in defect occurrences. We generally expect early defect occurrences to show a primarily increasing trend as users migrate to the release, exercise the software, and report defects, and we expect later defect occurrences to show a primarily decreasing trend as the rate of adoption declines and the release becomes more reliable due to defect removal.

The Weibull model has three parameters N, α, and β. Intuitively, the Weibull model can be broken down into three interacting pieces: N, representing the total number of defect

Table 1. Candidate models

Model type	*Model name*	*Model form*	*Researchers/users of the model*
Exponential	Non-homogenous Poisson process model	$\lambda(t) = N \alpha e^{-\alpha t}$	Goel & Okumoto
Weibull	Weibull	$\lambda(t) = N \alpha \beta t^{\alpha-1} e^{-\beta t^{\alpha}}$	Schick-Wolverton
Gamma	S-shaped reliability growth model	$\lambda(t) = N \beta^{\alpha} t^{\alpha-1} e^{-\beta t}$	Yamada, Ohba & Osaki
Power	Duane Model	$\lambda(t) = \alpha \beta e^{-\beta t}$	Duane
Logarithmic	Musa-Okumoto logarithmic Poisson model	$\lambda(t) = \alpha (\alpha \beta t + 1)^{-1}$	Musa-Okumoto

occurrences in the lifetime of the release, a generally increasing component $\alpha \beta t^{\alpha-1}$, which dominates early, and a decreasing component $e^{-\beta t^{\alpha}}$, which dominates as time increases.

N can be different for each release, which can account for differences in the total number of defect occurrences between releases. The differences may be caused by changes in software content or development processes.

In general, $\alpha \beta t^{\alpha-1}$ increases as a function of time and can account for increases in defect occurrences. The rate of growth is controlled by a combination of the α and β parameters. The increasing component is flexible enough to describe both concave and convex increasing patterns. Concave increasing patterns can occur when the growth in the rate of defect occurrences is faster at the beginning of the release, which may occur if many users quickly adopt and use a release. Convex increasing patterns can occur when the rate of defect occurrences increase slowly. This may occur if users slowly migrate to the release or if constraints on development and problematic content cause blocking, which prevents defects from being discovered.

The term $e^{-\beta t^{\alpha}}$ decreases as a function of time and can account for decreases in defect occurrences. Again, the rate of decrease is controlled by a combination of the α and β parameters. The decreasing component can describe concave or convex decreasing patterns. Convex decreasing patterns can occur when the rate of defect occurrences decrease rapidly, which may occur if there is fast migration to a new release. Concave patterns can occur when the rate of defect occurrences decrease slowly, which may occur if defect occurrences remain high over a longer time period due to constraints on development or problematic content.

It is reasonable to expect a model with parameterization that can describe varying defect-occurrence patterns such as the Weibull model to be better than other candidate models. The Exponential, the Power, and the Logarithmic models do not have both decreasing and increasing components. They cannot describe the interplay of increasing and decreasing trends. Although the Gamma model has both increasing and decreasing components, its decreasing component is generally convex. Thus, the Gamma model is unable to describe situations in which the decreasing pattern is concave. Kenny [11] has studied the interaction of increasing and decreasing trends in defect-occurrence patterns in commercial software systems and recommends using the Weibull model to model defect-occurrence patterns.

We hypothesize that the Weibull model is better than other candidate models at modeling defect-occurrence patterns for multiple releases of WPSSs.

3.3 Model fitting and model selection

We fit the best set of parameters for each candidate model for each release using Non-linear Least Squares (NLS) regression then compare the candidate models using the Akaike Information Criterion (AIC) model selection criterion [27].

NLS is a well-established model fitting procedure that selects model parameters by minimizing the square of the difference between fitted values and actual values [27]. It is widely used in defect modeling research [9][29]. We use the open source statistical computing package R [24]. After we select the best parameters for each candidate model for a given release, we use the AIC model selection criterion to evaluate the fit of the different candidate models; lower AIC scores are better. The AIC score is defined as:

$$AIC = n \log \sigma^2 + 2 |S|$$

where σ^2 is the residual squared error divided by the difference of the number of observations, n, and the number of model parameters, S [27]. The AIC model selection criterion penalizes models with more parameters to offset the advantage models with more parameters have in comparisons.

Our hypothesis stated in Section 3.2 would be supported if the Weibull model consistently produces lower AIC score.

Tables 2-5 present the AIC scores. The best AIC scores for each release are highlighted (shaded cells highlight the best AIC scores among fitted candidate models). INF and Singular Gradient indicate that the model-fitting algorithm is unable to fit parameters for the model, which suggests that the model is inappropriate. A detailed explanation of the failure to fit model parameters and why it suggests a model is inappropriate is given in [15, Appendix B].

Table 2. AIC scores for commercial OS

Release/ Model	*Exponential Model*	*Weibull Model*	*Gamma Model*	*Power Model*	*Logarithmic Model*
i	158	**131**	144	164	160
i + 1	119	**110**	111	131	126
i + 2	159	**150**	155	175	169
i + 3	**104**	105	111	113	109
i + 4	116	111	**109**	121	118
i + 5	104	**87**	89	105	104
i + 6	**62**	64	64	**62**	**62**
i + 7	Singular Gradient	**63**	**63**	66	Singular Gradient

Table 3. AIC scores for commercial middleware system

Release/ Model	*Exponential Model*	*Weibull Model*	*Gamma Model*	*Power Model*	*Logarithmic Model*
i	153	**134**	**134**	163	157
i + 1	Singular Gradient	173	**171**	195	Singular Gradient
i + 2	Singular Gradient	**116**	**116**	129	Singular Gradient

Table 4. AIC scores for open source OS

Release \Model	*Exponential Model*	*Weibull Model*	*Gamma Model*	*Power Model*	*Logarithmic Model*
2.6	167	**124**	129	173	170
2.7	162	**91**	103	176	171
2.8	182	**134**	136	188	185
2.9	135	97	**88**	143	139
3.0	110	**83**	90	113	112
3.1	170	168	**166**	173	171
3.2	88	**84**	85	101	99
3.3	91	**73**	76	93	91

Table 5. AIC scores for open source middleware system

Release /Model	*Exponential Model*	*Weibull Model*	*Gamma Model*	*Power Model*	*Logarithmic Model*
3.3	INF	**157**	**157**	193	192
4.0	INF	**215**	219	275	275
4.1	INF	**77**	78	92	93

Figure 1 shows fitted candidate models for a representative sample of releases from each of the four WPSSs. Due to confidentiality agreements, only the figures for open source software systems have numbered axes. Plots of all fitted candidate models are in [15, Appendix A].

Tables 2-5 show that the Weibull model has the best AIC score or one of the best AIC scores in 16 out of 22 or 73% of the releases. This is considerably better than the next best model, which is the Gamma model with the best AIC score or one of the best AIC scores in 8 out of 22 or 36% of the releases. Furthermore, since the AIC is a measure of deviance, it roughly follows a $\chi 2$ (chi-squared) distribution, which makes 4 a rough 95% confidence band around an AIC score. We note that the Weibull is within the 95% confidence band of the best AIC score in all but one of the releases. These results support our hypothesis that the Weibull model is the preferred model.

3.4 Validation of the Weibull model

We have shown that the Weibull is better than other candidate models. However, we still need to show that the Weibull model adequately describes the defect-occurrence pattern. We use the Theil forecasting statistic to validate the Weibull model.

The Theil statistic compares the forecast for each time interval *i* against a no-change forecast based on the previous time interval's value [26].

$$U^2 = \frac{\Sigma (P_i - A_i)^2}{\Sigma A_i^2}$$

The Theil statistic U is greater or equal to zero. The term P_i is the projected change and A_i is the actual change in interval i. A Theil statistic of zero indicates perfect forecasts with $P_i = A_i$. A Theil statistic of one indicates that forecasts are no better than no-change forecasts with $P_i = 0$. Values greater than one indicate forecasts are worse than no-change forecasts.

The Theil statistics shown in Figure 2 indicate that the best-fit Weibull is always better than the no-change forecast.

We conclude that the Weibull model is the preferred model because the Weibull model is better than other candidate models based on AIC scores in Section 3.3 and is good at projecting defect occurrences based on Theil statistics. Further validation is in [15, Appendix F].

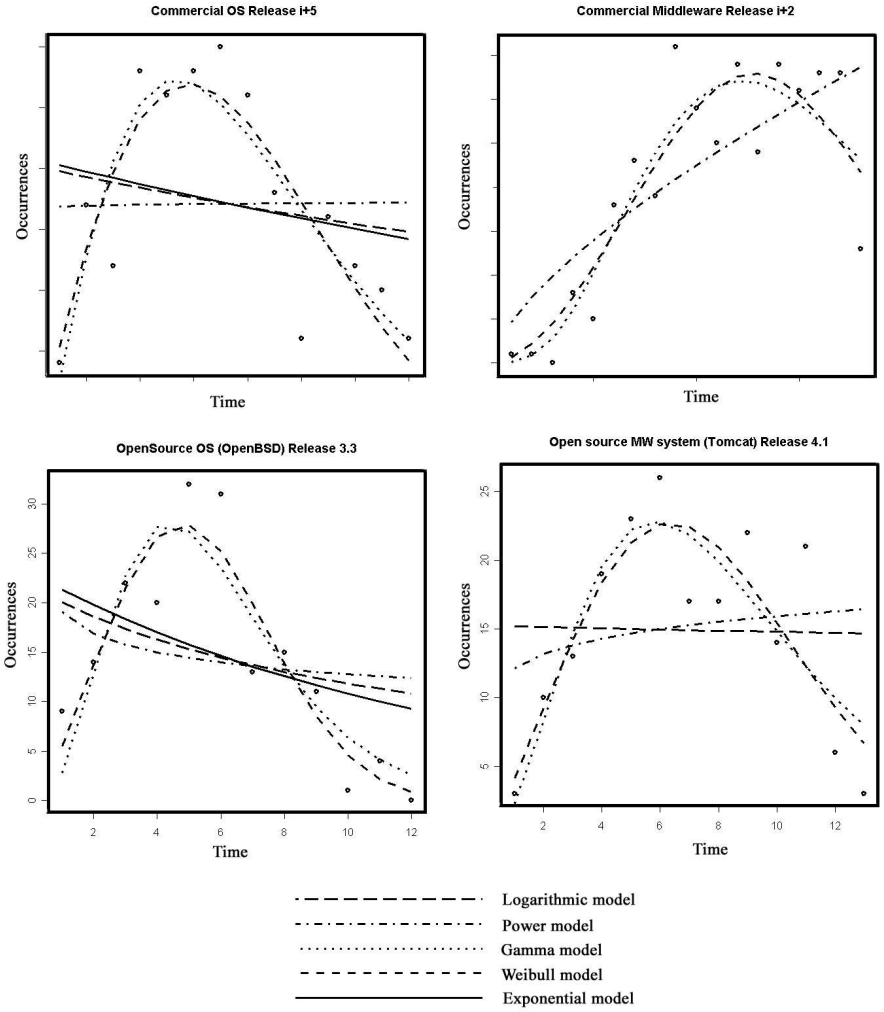

Figure 1. Samples of fitted candidate models

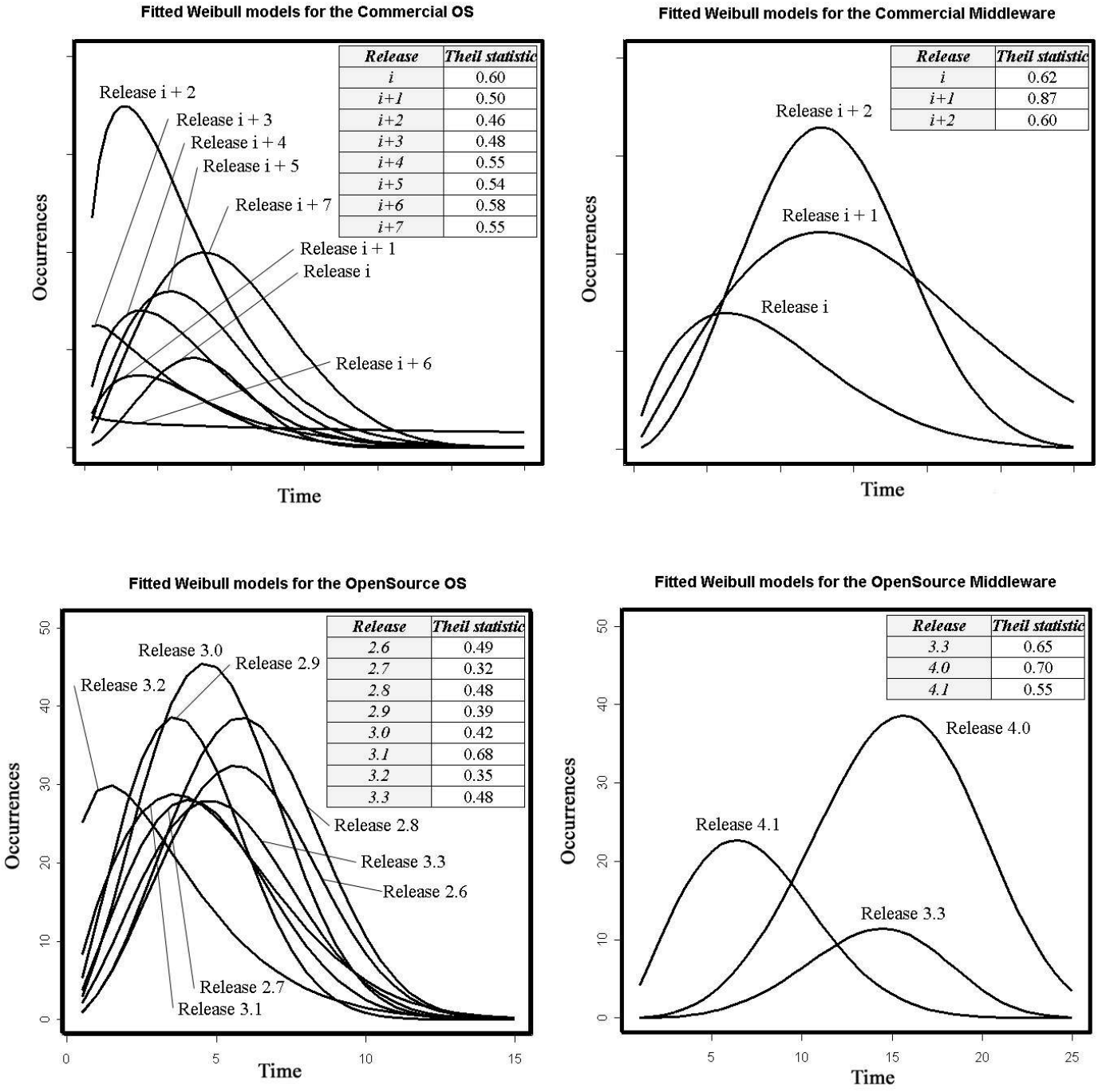

Figure 2. Plots of best-fit Weibull models

4. NAÏVE MODEL PARAMETER EXTRAPOLATION METHODS

Section 3.4 establishes the Weibull model as the preferred model. However, in order to project defect occurrences we still need to extrapolate model parameters for each new release. Naïve parameter-extrapolation methods that do not consider the changes in characteristics described in section 2.3 may extrapolate parameters that result in poor forecasts. In this section, we develop and empirically test the hypothesis that the moving averages and exponential smoothing methods are inadequate.

4.1 The moving averages method and the exponential smoothing method

Both the moving averages and exponential methods are well-established time series methods that represent intuitive, reasonable, and fairly common methods for extrapolating model parameters. The moving averages method extrapolates parameters by taking the average of the best-fit model parameters from the previous *k* releases. Exponential smoothing is similar, except more recent releases are given more weight, since intuitively more recent releases should be better predictors of the current release. Eick et. al. [6] have used a similar method to predict software defect rates for various software modules.

4.2 Inadequacies of naïve methods

Naïve parameter-extrapolation methods may not account for variations in defect-occurrence patterns because the methods do not consider changes in content, development process, deployment and usage patterns, and software and hardware configurations in use between releases. For example, consider a major release and a minor release. A major release that implements new functionality can have significant changes in content, production, and usage. New functionality can lead to substantial code changes and additions. If the schedule is fixed or if there is an insufficient number of trained testers then development constraints may cause inadequate testing. Finally, there may be many more adopters if the functionality being implemented is important to the users. A minor release that offers only minimal improvements over a previous release would have none of the above-mentioned conditions. We expect defect-occurrence patterns in the two releases to be drastically different. However, naïve parameter-extrapolation methods do not account for the differences.

We hypothesize that the two naïve parameter-extrapolation methods extrapolate model parameters that produce inadequate defect-occurrence projections for multiple releases of WPSSs.

4.3 Model parameter extrapolation and forecast evaluation

We evaluate naïve parameter-extrapolation methods by extrapolating model parameters and then examining the Theil forecasting statistics of the projected defect occurrences. Our hypothesis in section 4.2 will be supported if the two naïve parameter-extrapolation methods fail to consistently produce Theil statistics that are less than one and if the Theil statistics do not consistently improve with more data.

We increase the validity of our results by making the simplifying assumption that the total number of defect occurrences is known. Prior research has already shown that two different software releases are likely to have different total numbers of defect occurrences [5][17]. We remove the possible confound by providing estimates of the total number of defects. We show that naïve parameter-extrapolation methods are inadequate even with this simplification. This topic is discussed in detail in [15, Appendix D].

Tables 6-9 present fitted α and β parameters and Figure 2 presents plots of the best-fit Weibull models. We theorize that changes in characteristics described in Section 2.3 cause the variation in the parameter values.

Table 10 and 11 present Theil statistics resulting from using the moving averages and exponential smoothing methods to select the α and β model parameters of the Weibull model. The total number of defects is assumed to be given and is approximated by the total number of defects generated by the best-fit Weibull model.

We evaluate the benefits of including more historical information for naïve parameter-extrapolation methods by comparing the Theil statistics produced using multiple releases against the Theil statistics produced using model parameters of the most recent release as the model parameters for a new release. Improvements are possible since Theil statistics in tables 10 and 11 indicate that in 9 out of 18 or 50% of the forecasting experiments, using model parameters from the most recent release result in projections that are no better than no-change forecasts. (The exponential smoothing and moving averages methods are identical when extrapolating parameters using only data from the most recent release, thus their results are the same.) However, results show that projections do not improve if we incorporate additional historical information. There are 88 total forecasting experiments for the two naïve parameter-extrapolation methods using data from two or more releases. In 44 out of 88 or 50% of the forecasting experiments, Theil statistics show no improvement over values from column one release (shaded cells in tables 10 and 11 highlight values showing no improvement). Incorporating additional historical information failed to improve Theil statistics.

Table 6. Model parameters for commercial OS

Param /Rel	*i*	*i+1*	*i+2*	*i+3*	*i+4*	*i+5*	*i+6*	*i+7*
α	2.58	1.56	1.38	1.11	1.57	1.90	0.89	2.12
β	9.00	7.07	6.91	6.45	7.37	8.63	131.37	10.97

Table 7. Model parameters for commercial middleware system

Parameter /Release	*i*	*i+1*	*i+2*
α	1.72	2.05	2.81
β	10.58	17.75	14.97

Table 8. Model parameters for open source OS

Param /Rel	*2.6*	*2.7*	*2.8*	*2.9*	*3.0*	*3.1*	*3.2*	*3.3*
α	2.70	2.22	2.79	2.28	2.51	1.86	1.37	2.40
β	6.69	5.33	6.83	4.66	5.69	5.45	3.65	5.99

Table 9. Model parameters for open source middleware system

Parameter /Release	*3.3*	*4.0*	*4.1*
α	4.19	3.84	2.21
β	15.44	16.89	8.39

Table 10. Theil forecasting statistics for projections with the moving averages method

Releases/ System	*one release*	*two releases*	*three releases*	*four releases*	*five releases*	*six releases*	*seven releases*
Commercial MW R i+1	2.26						
Commercial MW R i+2	1.00	**1.11**					
Open source MW R4.0	0.94						
Open source MW R4.1	2.07	**2.07**					
Commercial OS R i+1	0.98						
Commercial OS R i+2	0.50	**0.88**					
Commercial OS R i+3	0.56	**0.61**	**0.86**				
Commercial OS R i+4	0.74	0.66	0.61	0.56			
Commercial OS R i+5	0.94	**1.30**	**1.29**	**1.22**	0.89		
Commercial OS R i+6	7.33	**7.44**	**7.43**	**7.46**	**7.52**	**7.63**	
Commercial OS R i+7	3.67	**3.76**	3.54	3.21	2.98	2.79	2.74
Open source OS R2.7	1.13						
Open source OS R2.8	1.06	0.70					
Open source OS R2.9	1.32	0.93	1.04				
Open source OS R3.0	0.87	0.42	0.43	0.44			
Open source OS R3.1	0.72	0.70	**0.73**	0.71	**0.73**		
Open source OS R3.2	0.76	**0.91**	**0.87**	**0.99**	**0.97**	**1.02**	
Open source OS R3.3	1.56	1.10	0.85	0.86	0.66	0.66	0.57

Table 11. Theil forecasting statistics for projections with the exponential smoothing method

Releases/ System	*one release*	*two releases*	*three releases*	*four releases*	*five releases*	*six releases*	*seven releases*
Commercial MW R i+1	2.26						
Commercial MW R i+2	1.00	**1.05**					
Open source MW R4.0	0.94						
Open source MW R4.1	2.07	**2.07**					
Commercial OS R i+1	0.98						
Commercial OS R i+2	0.50	**0.81**					
Commercial OS R i+3	0.56	**0.60**	**0.77**				
Commercial OS R i+4	0.74	0.67	0.63	0.57			
Commercial OS R i+5	0.94	**1.24**	**1.24**	**1.21**	**1.06**		
Commercial OS R i+6	7.33	**7.42**	**7.41**	**7.43**	**7.46**	**7.50**	
Commercial OS R i+7	3.67	**3.77**	**3.68**	3.56	3.50	3.47	3.48
Open source OS R2.7	1.13						
Open source OS R2.8	1.06	0.76					
Open source OS R2.9	1.32	1.00	1.06				
Open source OS R3.0	0.87	0.43	0.44	0.42			
Open source OS R3.1	0.72	0.70	**0.72**	0.71	**0.72**		
Open source OS R3.2	0.76	**0.88**	**0.86**	**0.93**	**0.93**	**0.95**	
Open source OS R3.3	1.56	1.18	0.99	0.98	0.87	0.86	0.82

Not only do naïve parameter-extrapolation methods fail to improve projections with additional data, they produce poor projections overall. Theil statistics are greater than or equal to one in 39 out of 88 or 44% of the forecasting experiments. Moving averages produced poor projections in 43% of the forecasting experiments. Exponential smoothing produced poor projections in 45% of the forecasting experiments. Similar results are produced when we use the naïve parameter-extrapolation methods to extrapolated model parameters for the Gamma model. The details are in [15, Appendix F].

We conclude that there is strong empirical evidence that the Weibull model is the preferred model for modeling defect-occurrence patterns of WPSSs across multiple releases and that the naïve parameter-extrapolation methods, moving averages and exponential smoothing, are inadequate in extrapolating model parameters of the Weibull model for defect-occurrence projection.

5. VALIDATION AND FUTURE WORK

Our research aims to deal with the real world consequences of defect occurrences in WPSSs. Maintenance planning and software insurance are two methods that can deal with the consequences [13], and both need accurate defect-occurrence projections. This paper set out to address two questions that are important for defect-occurrence projection: is there a type of defect model that provides a good fit to defect-occurrence patterns across multiple releases and in many organizations, and how can model parameters for a new release be extrapolated using historical information.

We have examined historically effective defect-occurrence models. There is extensive defect-occurrence modeling research history to support our belief that our collection of models is well-suited for modeling software defect-occurrence patterns. Despite these efforts, it is possible that a better defect model exists, and we strongly encourage others to replicate our approach using a wider array of models.

The difference in definition of defect occurrences and the time interval used between the software systems strengthens our finding that the Weibull model is the preferred model, since despite the differences in definition, the Weibull model still proved superior. There may be other meaningful ways of counting defects where the Weibull model does not perform as well. Only future research can address this issue.

We have attempted to establish the external validity by including two different types of software systems (middleware and

operating systems) developed with two different development styles (commercial and open source). It is not yet clear how adequately this sample represents the population of WPSSs. We regard this as a promising start, but future research should sample additional parts of the WPSS space.

The goal of our research is to develop a defect-occurrence projection method that produces defect-occurrence projections that are better than post-facto fits. Our results show that naïve parameter-extrapolation methods are never better than the post-facto, best-fit, Weibull model (Theil statistics in Figure 2 and Tables 10 and 11). However, a post-facto fit chooses model parameters that minimize the residual error for all data points simultaneously, which does not necessarily produce the best model parameters for each data point. It may be possible to produce better projections than the post-facto fit by updating a-priori projections as more information becomes available, such as after the arrival of field defect-occurrence data or after the release of software patches.

Our results in this paper indicate that the Weibull model is the preferred model for modeling defect-occurrence patterns for multiple releases of WPSSs and that the naïve parameter-extrapolation methods are inadequate. We have claimed that we expected the naïve parameter-extrapolation methods to fail because they do not account for differences in content, development process, deployment and usage patterns, and software and hardware configurations in use. The next step should be to improve parameter-extrapolation methods by predicting the effects on model parameters resulting from changes in characteristics of widely-deployed, production, software systems.

6. ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation under Grand CCR-0086003, by the Sloan Software Industry Center at Carnegie Mellon University, and by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298. Thanks to IBM for making this work possible, Larry Wasserman for his guidance, and Audris Mockus for his insights.

7. REFERENCES

[1] K. Bassin and P. Santhanam. Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. In *Eighth International Symposium on Software Reliability Engineering, Case Studies*, p103-114, 1997.

[2] B. Boehm and et. al. Cost models for future software life cycle processes: COCOMO 2.0. In *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, Chapter 1,1995.

[3] M. Buckley and R. Chillarege. Discovering relationships between service and customer satisfaction. In *The International Conference on Software Maintenance*, p192-200 1995.

[4] R. Chillarege, S. Biyani, and J. Rosenthal. Measurement of failure rate in widely distributed software. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing*, p424-433, 1995.

[5] S. Chulani. COQUALMO (constructive quality model) a software defect density prediction model. In *Project Control for Software Quality*, 1999.

[6] S. G. Eick, T. L. Graves, A. F. Karr, J. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. In *IEEE Transactions on Software Engineering*, Volume 27, p1-12, 2001.

[7] D.E. Harter, M.S. Krishnan, S.A. Slaughter. Effects of process maturity on quality, cycle time, and effort in software product development. *Management Science,* Volume 46, p451-466, 2000.

[8] Jakarta Tomcat. http://jakarta.apache.org/tomcat/index.html

[9] W. Jones. Reliability models for very large software systems in industry. In *International Symposium Software Reliability Engineering*, p17-18, 1991.

[10] W. Jones, J. Hudepohl, T. M. Khoshgoftaar, and E. B. Allen. Application of a usage profile in software quality models. In *Third European Conference on Software Maintenance and Reengineering*, p148-157, 1999.

[11] G. Kenny. Estimating defects in commercial software during operational use. In *IEEE Transactions on Reliability*, Volume 42, p107-115, 1993.

[12] M. Lehman and L. Belady. *Program Evolution: Processes of Software Change*. Academic Press, USA, 1985.

[13] P. Li, M. Shaw, and J. Herbsleb. Selecting a defect prediction model for maintenance resource planning and software insurance. In *EDSER-5 affiliated with ICSE*, p32-37, 2003.

[14] P. Li, M. Shaw, K. Stolarick, and K. Wallnau. The potential for synergy between certification and insurance. In *International Workshop on Reuse Economics in conjunction with ICSR*, 2002.

[15] P.Li, M.Shaw, J.Herbsleb, B. Ray, and P.Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. In *CMU tech report CMU-ISRI-04-130*, 2004

[16] M. R. Lyu. *Handbook of Software Reliability Engineering*. IEEE Society Press, USA, 1996.

[17] A. Mockus, D. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *ICSE*, p274-284, 2003.

[18] P. Mora and Z. Jelinski. Final report on software reliability study. In *McDonell Douglas Astronautic Company Report Number 63921*, 1972.

[19] J. Musa. A theory of software reliability and its applications. In IEEE Transaction on Software Engineering, Volume 3, p312-327, 1975.

[20] J. Musa. Operational profiles in software reliability engineering. In *IEEE Software*, Volume 10, p14-32, 1993.

[21] OpenBSD. http://www.openbsd.org

[22] T. Ostrand and E. Weyuker. The distribution of faults in a large industrial software system. In *ISSTA*, p55-64, 2002

[23] K.Popstajanova and K. Trivedi. Architecture based approach to reliability assessment of software systems. *Performance Evaluation*, Volume 45, 2001.

[24] R project for statistical computing. http://www.r-project.org

[25] E. Raymond. *Cathedral and the Bazaar*. O'Reily & Associates, USA, 1999.

[26] H. Theil. *Applied Economic Forecasting*. North-Holland Publishing Company, Netherlands, 1966.

[27] W. Venables and B. Ripley. *Modern Applied Statistics with S-plus*. Springer Verlag, USA, 1996.

[28] J. Voas. User participation-based software certification. In *Eurovav*, p267-276, 1999.

[29] A. Wood. Predicting software reliability. In *IEEE Computer*, Volume 9, p69-77, 1999.