RESEARCH FEATURE

Selecting the Best Reliability Model to Predict Residual Defects in Open Source Software

Najeeb Ullah and Maurizio Morisio, Polytechnic University of Turin
Antonio Vetrò, Technical University of Munich

A proposed method evaluates eight popular software reliability growth models and selects the one that can best predict the software's remaining faults, providing practical support for project managers who are considering an open source component.

Software reliability—the probability of failure-free software operations for an extended period in a specific environment[1]—is a critical quality characteristic that affects both safety and cost. For open source software (OSS), for example, reliability is foundational to widespread adoption. Consequently, decades of research have focused on developing and refining various classes of software reliability models, one of which is the software reliability growth model (SRGM), which testers use to determine the cumulative number of expected defects in software before release or, less commonly, to predict its residual defects—the remaining faults or failures after release.

As the sidebar "Software Reliability Growth Model Basics" describes, each SRGM is a mathematical expression that specifies the general form of the software failure process as a function of factors—such as fault introduction and removal—and the operational environment.[1] The model assumes that the failure rate (failures per unit of time) generally decreases as testers identify and remove defects.

An SRGM estimates the failure rate's curve shape by statistically estimating certain parameters, which are specific to the chosen SRGM. With this shape, testers can estimate the extra time required to meet a specified reliability objective and identify the software's expected reliability after release.[1] However, because each application has its own failure-rate curve, no universally applicable SRGM is possible. Moreover, there is no consensus on how to select the best model, and no studies have yet proposed an empirical SRGM selection methodology that is suitable for OSS, which has different needs than software developed in house.

To address that gap, we developed a method to select the best SRGM among several candidates specifically for predicting residual defects in OSS—a measure of concern to project managers who must decide whether or not to include an OSS component. We tested our method empirically by applying 8 popular SRGMs to 21 releases of 7 OSS projects. We also quadrupled the number of

 0018-9162/15/$31.00

SOFTWARE RELIABILITY GROWTH MODEL BASICS

Software reliability growth models (SRGMs) have either a concave or an S-shaped curve. S-shaped models assume that the cumulative failures occur in an S-shaped pattern. Testers are initially unfamiliar with the product, and the fault removal rate slowly increases as they become more familiar. As the testers' skills further improve, that rate increases more quickly and then levels off as residual defects become more difficult to find. The concave models do not include an initial learning curve. Rather, they assume that the failure-rate increase reaches a peak and then becomes stable.

All SRGMs use a nonhomogeneous Poisson process (NHPP) to model the failure process, which is characterized by its mean value function (MVF). If $\{N(t), t > 0\}$ denotes a counting process that represents the cumulative defects detected in t, then an SRGM based on an NHPP is defined as[1]

$$P\{N(t) = n\} = \frac{m(t)^n}{n!} e^{-m(t)}, n = 1, 2 \ldots,$$

where MVF is represented as $m(t)$ and is nondecreasing in t under the bounded condition $m(\infty) = a$, with a being the expected total number of defects that testers will eventually detect.

The value of a tells testers whether the software is ready for release or how much additional testing is required if it is not ready. Varying the MVF makes it possible to define different NHPP models.

More basic details on SRGM use are available at the Polytechnic University of Turin website (http://softeng.polito.it/najeeb/IEEE/QuickRefresher.pdf).

Reference

1. M.R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

datasets others have used for the evaluation, allowing us to observe our proposed method's output across a variety of projects and across releases of the same project, each with different defect data amounts.

Our method is different from other approaches because it emphasizes both OSS and the prediction of residual defects, not the cumulative defect expected number. Thus, it is suitable for stable released OSS projects that do not require formal testing. Unlike other methods, our approach is also suitable for practitioners with no background in statistics.

An empirical validation shows that our method is highly effective. Of the 21 releases, it chose the model with the highest estimation precision in 17 releases and with the second highest precision in the other 4 cases.

MODEL COMPARISON METHODS

With no universal SRGM, project managers need some way to choose the best model from myriad available candidates. Unfortunately, despite decades of research, the only consensus about selection is that it should be done on a case-by-case basis. There is no universally accepted selection criterion or metric, and the selection criteria that have been reported were evaluated on very few projects. Some work looks at why selection is difficult, rather than focusing on a selection method. One research group[2] observed that hidden design flaws are the main causes of software failures, which makes model selection problematic.

Models also differ in their intended application. Some SRGM applications look at the total number of cumulative defects at some point in time, which is evident when reliability starts to stabilize. Other SRGM applications are more interested in predicting the total number of defects that will eventually occur and, by extension, the residual defects. Although most research focuses on the former perspective, we believe that the latter perspective characterizes software reliability more concretely.

Metrics versus selection method

In the literature we reviewed, researchers typically applied comparison metrics to some number of SRGMs and noted patterns. Only Catherine Stringfellow and A. Amschler Andrews of Midwestern State University[3] proposed a selection method. Like our method, theirs attempts to predict residual defects, but Stringfellow and Andrews validated their method only on closed source software (CSS) for which testers have completed 60 percent of the planned tests.

Although Stringfellow and Andrews provided no guidelines for adapting their method to OSS, it can help testers decide whether or not to stop testing and release CSS.

Predictive quality

In cumulative defect prediction, many researchers have shown that some model types have higher predictive quality; for example, geometric models—those based on hypergeometric distribution—have better predictive quality than other models.[4] One research group in a different study found that different models work well only on certain datasets, so comparing models' predictive quality for a given application is the best selection approach.[5]

Another group of researchers, which analyzed the predictive quality of 10 models using 5 metrics, observed that the best predictive model depends on the metric used, since the different metrics in their study produced different model choices for the same dataset.[6] Two other approaches rank different models in terms of best fit but do not select the best predictor.[7,8]

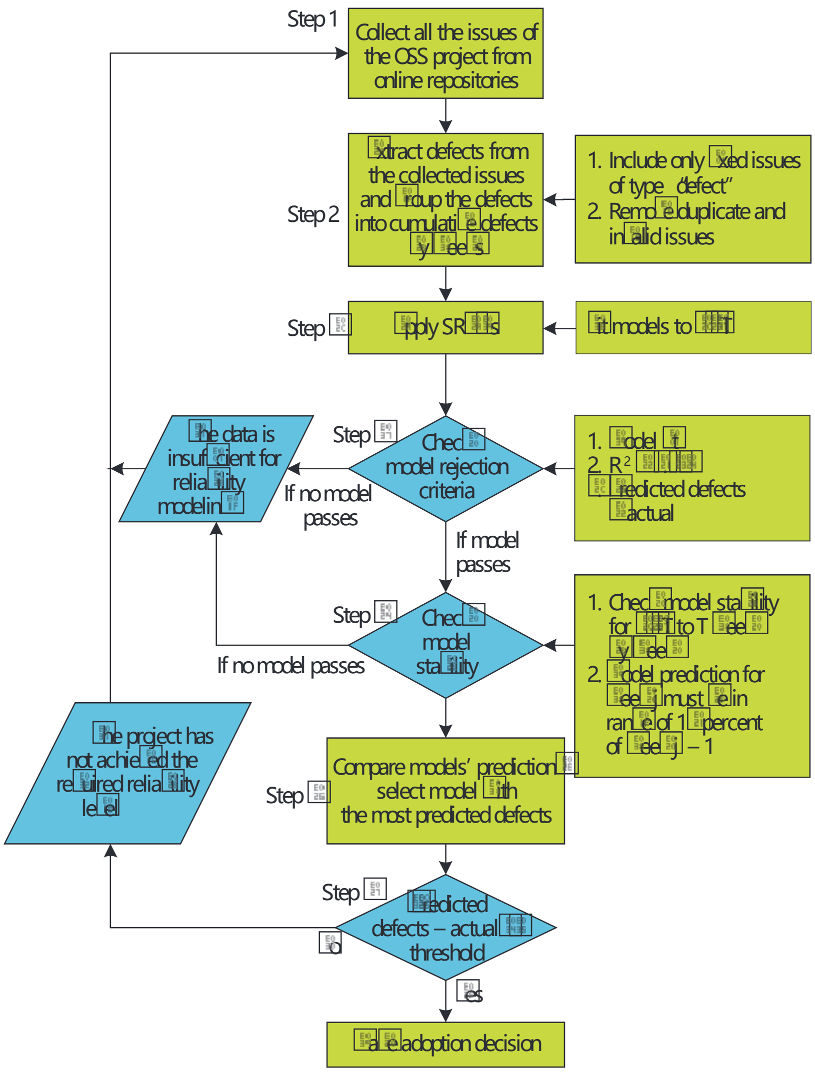

FIGURE 1. Steps in the proposed method of selecting the software reliability growth model (SRGM) with the best fit to data and the best predictive quality in estimating expected residual defects in open source software (OSS).

Of the work we reviewed, all researchers evaluated the predictive quality of SRGMs on the basis of fitting the models to one portion of the defect dataset and predicting only the second portion. Except for the study by Stringfellow and Andrews, all these efforts evaluated a model's predictive quality only in terms of the software's overall behavior, not its residual defects. Evaluations based on overall behavior show only which model outperforms the others, which is not useful insight for practitioners. These studies also used only five or fewer datasets to validate their methods.

METHOD OVERVIEW

We designed our method to select the SRGM that has the best fit to an OSS component's defect dataset and is the best predictor of the component's total number of expected residual defects. Our main purpose is to support project managers in deciding whether or not to adopt an OSS component.

Our method derives from Stringfellow and Andrews' work but must deal with two new problems that stem from the nature of OSS:

- Because many SRGMs have assumptions that might not apply to OSS, a particular model might not fit the data or have a low goodness of fit (GoF).
- The defect data for an OSS component is usually limited. The smaller the defect dataset, the longer it might take the models to stabilize.

To address these problems, our method uses several SRGMs and selects the models that best fit the data.

METHOD IMPLEMENTATION

Figure 1 shows a flow graph of the steps in our method.

Step 1: Collect defect data

After selecting the OSS release of interest, the first step is to collect issues and defect data from online repositories of the OSS project, which are SourceForge, Apache, and Bugzilla.

Step 2: Extract defects

The issues collected in step 1 can be bugs, feature requests, improvements, or tasks, so they must be filtered to remove all issues except bugs or defects—no enhancements, feature requests, tasks, or patches—and only defects reported as closed or resolved. Filtering should exclude open, reopened, or duplicate defects. Part of step 2 also involves grouping the defect data of the entire release interval [0,T] into cumulative defects by weeks. Figure 2 shows this interval graphically.

Step 3: Apply models to data

Table 1 lists the eight SRGMs that our method uses. However, these models are a particular context for applying the method. The remaining steps, as well as the release interval, are meant

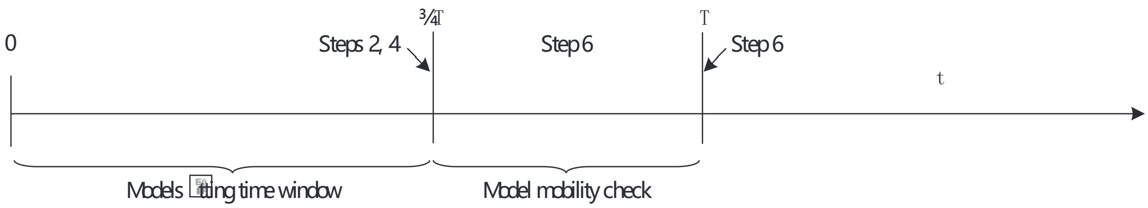

FIGURE 2. Timeline of method steps across two-thirds of an OSS release interval [0,T]. The other third of total release time t is for validation. Steps 3 through 4 represent the model fitting time window, which takes approximately three-fourths of the method application time, while step 5, the window for determining model stability, takes the other fourth. Steps 6 and 7, select model and compute residual defects, occur at the interval's end.

TABLE 1. Software reliability growth models used in our method.

Model name	Type	Mean value function $m(t)$
Musa-Okumoto*	Concave	$m(t) = a\ln(1+bt),\ a>0,\ b>0$
Inflection S-shaped†	S-shaped	$m(t) = a\dfrac{1-\exp[-bt]}{1+\psi(r)\exp[-bt]},\ \psi(r)=\dfrac{1-r}{r},\ a>0,\ b>0,\ r>0$
Goel-Okumoto†	Concave	$m(t) = a(1-\exp[-bt]),\ a>0,\ b>0$
Delayed S-shaped†	S-shaped	$m(t) = a(1-(1+bt)\exp[-bt]),\ a>0,\ b>0$
Generalized Goel†	Concave	$m(t) = a(1-\exp[-bt^c]),\ a>0,\ b>0,\ c>0$
Gompertz†	S-shaped	$m(t) = ak^{b^t},\ a>0,\ 0<b<1,\ 0<k<1$
Logistic†	S-shaped	$m(t) = \dfrac{a}{1+k\exp[-bt]},\ a>0,\ b>0,\ k>0$
Yamada exponential‡	Concave	$m(t) = a(1-\exp(-r(1-\exp(-bt)),\ a>0,\ b>0,\ r>0$

*J.D. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," Proc. 7th Int'l Conf. Software Eng. (ICSE 84), 1984, pp. 230–238.
†M. Xie, Software Reliability Modeling, World Scientific Publishing, 1991.
‡H. Pham, "Software Reliability and Cost Models: Perspectives, Comparison and Practice," European J. Operational Research, vol. 149, 2003, pp. 475–489.

to be a generic description. In this step, testers apply the models to defect data from step 2, placing them in the model fitting window—marked with the 3/4T point in Figure 2 (designating three-fourths of the time allotted for method application).

Because of the nature of defect data, testers can use a general technique, nonlinear regression (NLR), to fit a model to the data. NLR estimates parameters by minimizing the sum of the squares of the distances between the data points and the regression curve. It is an iterative process that starts with an initial estimated value for each parameter. The iterative algorithm then gradually adjusts these parameters until they converge on the best fit. Consequently, adjustments make virtually no difference in the sum of squares.

If the model cannot describe the data, its parameters cannot converge to the best fit, so there is no fit. If a fit is possible, our method evaluates the model's GoF on the basis of the R^2 value, which determines how well a curve fits the data. The value is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{k}\left(m_i - m(t_i)\right)^2}{\sum_{i=1}^{k}\left(m_i - \sum_{j=1}^{k}\frac{m_j}{n}\right)^2},$$

where k represents dataset size, $m(t_i)$

represents predicted cumulative failures, and m_i represents actual cumulative failures at time t_i.[9] R^2 takes a value between 0 and 1, inclusive. The closer the R^2 value is to one, the better the fit. We chose the R^2 value for its simplicity and because an evaluation of several statistical tests for GoF showed that this measure was at least as powerful as the other tests analyzed.[10]

The larger the R^2 value, the better the curve fits the data, and the easier it is to see any data variation. Fitting a model enables us to estimate the value for all model parameters, notably the expected number of total defects (the a parameter).

For model fitting, we used Prism, a commercial curve-fitting program that employs NLR techniques for curve fitting and supplies model equations and parameter constraints. The program then fits the model to the data and returns an estimate of the best-fitted values for all the parameters of the models along with the GoF value (R^2).

Step 4: Test against fit and prediction thresholds

The fitted models must pass a test based on a threshold GoF value. In this step, the method compares the fitted models' GoF values with the specified R^2 value threshold. The threshold is a subjective decision, and other applications of our method might have a different value. Our GoF value threshold of 0.95 is based on Stringfellow and Andrews's work.

This step also involves checking the fitted models' predictions against the actual number of defects found. Only the models whose prediction is greater than the actual number of defects pass the rejection test, as prediction is meaningless if the model predicts a number that is lower than the actual number of defects.

If no model passes this step, the collected defect data is insufficient for reliability modeling and all the models fail the test, or testers must supply additional data.

Step 5: Test against prediction stability threshold

In this step, the method evaluates the remaining models' prediction stability. (In our study, all models passed step 4, but other comparisons might have different results.) A prediction is stable if the prediction for week j is within ±10 percent of the prediction for week j − 1. Again, threshold setting is a subjective decision. We used a threshold of 10 percent because of its successful use in other work.[11] If no model has a stable prediction within the threshold, the collected defect data is insufficient for reliability modeling, and testers must supply additional data.

Our method checks model stability within the window of 3/4T to T—the model stability check window in Figure 2. The method checks stability in the same manner for all models: it adds one week of defects to the cumulative defects of 3/4T, so 3/4T + 1 week. It then fits all the models that have passed the rejection step to the cumulative defects in 3/4T + 1 week. Next, it adds another week, so 3/4T + 2 weeks, and so on until it reaches the total number of weeks in the release T.

Step 6: Select the best model

The best SRGM is the one that has passes all the threshold tests and has the highest number of predicted defects. The choice is conservative, but models that overestimate the actual number of defects will be more suitable for a project manager's goal, as cost estimates will be worst case, lowering the risk of adoption (no cost surprises). Widely differing prediction values among models might require conducting an additional analysis with other quality assessment methods,[12] or choosing a subset of models based on the GoF indicator.

Step 7: Compute residual defects

Once the method has selected the best SRGM, it uses that model to compute the OSS's residual defects. With this number, the project manager can decide whether to adopt the OSS, wait until more defects are identified and fixed, or evaluate a different OSS.

METHOD APPLICATION

We applied our method to a cross-section of OSS projects, selecting seven projects with varying natures and large, well-organized communities: Apache, Gnome, C++ Standard Library, JUDDI, HTTP Server, XML Beans, and Enterprise Social Messaging Environment (ESME).

Gathering project data

For Apache and Gnome, we took defect data on three releases each from a published report that had already grouped them into cumulative defects by weeks from release date.[13] We identified the other five open source projects from Apache.org (https://issues.apache.org), which characterized the projects as stable in production with reported issues fixed and closed (66, 95, 68, 64, and 82 percent of reported issues, respectively).

We used Atlassian's JIRA issue tracker (www.atlassian.com/software/jira) to collect defect data for the five projects. JIRA tracks bugs and tasks, links issues to related source code,

TABLE 2. Model fitting for stability check for Gnome release 2.0.

Weeks after release	Actual defects	Delayed S-shaped		Logistic		Gompertz		Generalized Goel	
		Pred*	R^2	Pred*	R^2	Pred*	R^2	Pred*	R^2
12	58	68	0.974	59	0.9937	73	0.9889	105	0.9819
13	58	71(S)	0.9781	62(S)	0.9937	74	0.9909	100	0.9852
14	66	78	0.9764	69	0.9879	84(D)	0.9891	202(D)	0.9866
15	72	86	0.9747	78	0.9844				
16	74	90	0.9772	83	0.986				

*Pred: total defects predicted

plans agile development, monitors activity, and reports project status.

We downloaded all the issues for 3 releases of C++ Standard Library, 3 releases of JUDDI, 2 releases of HTTP Server, 4 releases of XML Beans, and 3 releases of ESME—a total of 15 releases.

For each issue, JIRA records the project name and useful information, such as

- key, summary, and issue type—the unique identity of each issue, a comprehensive description, and whether the issue is a bug, task, improvement, or new feature request;
- status and resolution—the current status can be resolved, closed, open, or reopened, and the resolution can be fixed, duplicate, or invalid;
- created and updated or fixed dates and times; and
- affected versions—the project releases that had the issue.

Following step 2, we manually filtered JIRA data to include only closed or resolved issues and then filtered those results again to include only defects or bugs.[14] We then grouped the refined data into cumulative defects by weeks. The full defect dataset of each release is available at the Polytechnic University of Turin website (http://softeng.polito.it/najeeb/IEEE/datasets.pdf).

Results

Method application took about two-thirds of the time window in Figure 2, with the remaining third devoted to validation. For example, the time interval for Gnome release 2.0 is 24 weeks, so method application was across 16 weeks (2/3 × 24). Of the 16 weeks, we used the last 4 weeks for model stability checking. We chose two-thirds as a window for estimating model parameters because previous studies implied that model parameters do not become stable until about 60 percent of testing is complete.[11]

Table 2 shows partial results for the Gnome 2.0 release for weeks 12 to 16. The Musa, Inflection, Goel, and Yamada SRGMs destabilized in week 13. The table shows results for the remaining four models.

Columns from left to right show the number of actual cumulative defects found in that week and, for each SRGM shown, the number of total defects predicted (Pred) and the GoF value (R^2) value. The results for all project releases are available at the Polytechnic University of Turin website (http://softeng.polito.it/najeeb/IEEE/Remaining_results.pdf).

The method filed each SRGM each week, marking a model with R, S, or D to denote causes for elimination:

- R: failed the fitting and prediction check (did not occur in this release),
- S: model stabilized that week, and
- D: status changed to unstable (destabilized).

All the models—including the four not shown—performed well in fitting and passed the rejection test (step 4), but their predictive quality differed considerably. Five of the eight models destabilized by week 14, and all five significantly overestimated the defect number ("Actual defects" column in Table 2).

Table 2 also shows that the Delayed S-shaped and Logistic models stabilized first at week 13 and remained stable up to week 16 (throughout the entire stability check window). The Delayed S-shaped and Logistic models predicted the number of defects at week 16 as 90 and 83, respectively. The Delayed S-shaped model predicted more residual defects than the Logistic model did, so the method selected it as the best (underlined values in the table).

VALIDATION

For validation, the remaining third of the release interval, we measured each model's prediction capability using prediction relative error (PRE):

$$PRE = \frac{\text{Predicted - Actual number of defects}}{\text{Predicted}},$$

where Predicted is the total number of defects a model predicted at two-thirds of the time interval, and Actual is the

TABLE 3. Validation results in choosing the best predictor model.

Project	Release	Model selected by prediction relative error	Model selected by our method
Gnome	V2.0	Delayed S-shaped	Delayed S-shaped
	V2.2	Goel-Okumoto, Yamada exponential	Goel-Okumoto
	V2.4	Inflection S-shaped, Gompertz	Inflection S-shaped
Apache	2.0.35	Delayed S-shaped, Logistic	Gompertz
	2.0.36	Delayed S-shaped, Logistic, Gompertz, Generalized Goel	Generalized Goel
	2.0.39	Inflection S-shaped, Goel-Okumoto, Generalized Goel	Goel-Okumoto
C++ Standard Library	4.1.3	Musa-Okumoto	Inflection S-shaped
	4.2.3	Musa-Okumoto	Gompertz
	5.0.0	Inflection S-shaped, Yamada exponential, Generalized Goel	Yamada exponential
JUDDI	3.0	Musa-Okumoto, Goel-Okumoto, Delayed S-shaped, Yamada exponential, Gompertz	Delayed S-shaped
	3.0.1	Musa-Okumoto, Delayed S-shaped	Delayed S-shaped
	3.0.4	Goel-Okumoto	Goel-Okumoto
HTTP Server	3.2.7	Goel-Okumoto	Goel-Okumoto
	3.2.10	Delayed S-shaped, Logistic	Logistic
XML Beans	2.0	Gompertz	Delayed S-shaped, Gompertz
	2.2	Logistic	Logistic
	2.3	Musa-Okumoto, Goel-Okumoto, Logistic, Yamada exponential, Generalized Goel	Logistic
	2.4	Delayed S-shaped, Gompertz	Gompertz
Enterprise Social Messaging Environment	1.1	Musa-Okumoto, Goel-Okumoto	Goel-Okumoto
	1.2	Logistic, Gompertz	Gompertz
	1.3	Delayed S-shaped, Logistic, Gompertz	Inflection, Goel-Okumoto, Generalized Goel

number of defects at the end of the time interval.

For each release and each project, we computed the PRE for each model, ranked the models accordingly, and considered the model with the lowest PRE as the best predictor for that release.

Table 3 compares the best predictors chosen by PRE and by our selection method for each project release. For 17 of the 21 releases, the PRE and our method chose the same model. In the remaining four releases, the best model had a negative PRE, so our method rejected the model. Even so, the method still chose the second-best model (the one with the lowest positive PRE) in all four of the releases.

OBSERVATIONS

Our results are promising and prompt several observations that could guide future work.

Validity of results

With 21 datasets, our validation is more extensive than any similar study to date. However, because we used datasets that others produced, we could not control the quality of the issues collected and reported. Issues, time to fix, or description could have been missing. We tried to mitigate this lack of control by selecting established OSS projects with large communities and using datasets that similar research efforts have employed.

One validity threat is the lack of generalization. Although we used the largest number of datasets with the largest variety of release intervals, which vary from days to a year, we still cannot generalize our results to all project releases because a model could still fail to fit the data or fail the GoF or stability threshold tests.

Model ranking

Of the 21 datasets, no model ranked best in more than a few cases, and each of the 8 models was ranked best at least once. This result is consistent with related work and provides further evidence that any methodology must select the best SRGM on a project-by-project basis.

That said, models with certain characteristics might be better predictors. For example, in 14 of the 21 releases, the best model was S-shaped, not concave. Our previous studies[14] had similar results. One possibility is that the OSS project community members (users and reviewers) tend not to react immediately to a new release—behavior that is consistent with the learning phase in S-shaped models.

Model fit

In line with a common assumption in the literature, we considered each release to be its own independent project,[11,12,14,15] and our results show that different models fit different releases of the same projects. One explanation is that model selection is based only on defect history, not on any other project characteristic, and the factors that determine defect history are not well understood. Defects can stem from characteristics like code complexity or from the domain's inherent challenges, or even from the degree of coding skill. An organization's processes and management choices can also influence project quality.

Amount of defect data

Our method rejected models primarily because of prediction instability, not GoF value, which could signal the need for more defect data. The method overcomes this instability by 21 releases, but evaluating this much data might not be feasible with other applications. Additional research might focus on determining the minimum amount of defect data needed to select an SRGM.

Parameter choices

The thresholds and windows in our method seemed to perform well: GoF minimal fitting threshold of 0.95, stability threshold of 10 percent, three-fourths interval for method application, and one-fourth interval for fitting and stability check. A threshold sensitivity analysis was outside the scope of our study, but it could be a topic for future work.

Our work aims to support practitioners in characterizing OSS reliability in terms of residual defects, with the larger goal of helping project managers decide on OSS use. We believe that our method key contributions are its systematic approach and the extent of validation. In future work, we plan to refine our method and conduct another study to explore additional selection methods and possibly develop supporting tools to automate aspects of our method.

ABOUT THE AUTHORS

NAJEEB ULLAH is a PhD candidate in computer engineering at Polytechnic University of Turin, Italy. His research interests include software reliability, software process improvement, and empirical software engineering. Ullah received an MS in computer engineering from Polytechnic University of Turin. He is a member of IEEE. Contact him at najeebullahkhan1984@gmail.com.

MAURIZIO MORISIO is a professor and Software Engineering Group leader in the Department of Control and Computer Engineering at Polytechnic University of Turin. His research interests include empirical software engineering, software processes, and green IT. Morisio received a PhD in software engineering from Polytechnic University of Turin. He is a member of IEEE and associate editor in chief of IEEE Software. Contact him at maurizio.morisio@polito.it.

ANTONIO VETRÒ is a postdoctoral research fellow in the Software and System Engineering Department at the Technical University of Munich, Germany. His research interests include empirical methodologies, analyses of process and product data for software quality, and data quality assessment. He is a member of ACM. Contact him at vetro@in.tum.de or phisaz@gmail.com.

REFERENCES

1. IEEE Std. 1633-2008, IEEE Recommended Practice on Software Reliability, IEEE, 2008; http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4554206.
2. S. Brocklehurst et al., "Recalibrating Software Reliability Models," IEEE Trans. Software Eng., vol. 16, no. 4, 1990, pp. 458–470.
3. C. Stringfellow and A.A. Andrews, "An Empirical Method for Selecting Software Reliability Growth Models," Empirical Software Eng., vol. 7, no. 4, 2002, pp. 319–343.
4. J.D. Musa, A. Iannino, and K. Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill College, 1987.
5. A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and

Applicability," IEEE Trans. Software Eng., vol. 11, no. 12, 1985, pp. 1411–1423.

6. A.A. Abdel-Ghaly, P.Y. Chan, and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," IEEE Trans. Software Eng., vol. 12, no. 9, 1986, pp. 950–967.
7. N. Miglani and P. Rana, "Ranking of Software Reliability Growth Models Using Greedy Approach," Global J. Business Management and Information Tech., vol. 1, no. 11, 2011, pp. 119–124.
8. M. Anjum, A. Haque, and N. Ahmad, "Analysis and Ranking of Software Reliability Models Based on Weighted Criteria Value," Int'l J. Information Technology and Computer Science, vol. 5, no. 2, 2013, pp. 1–14.
9. K.C. Chiu, Y.S. Huang, and T.Z. Lee, "A Study of Software Reliability Growth from the Perspective of Learning Effects," Reliability Eng. & System Safety, vol. 93, no. 10, 2008, pp. 1410–1421.
10. O. Gaudoin, B. Yang, and M. Xie, "A Simple Goodness-of-Fit Test for the Power-Law Process based on the Duane Plot," IEEE Trans. Reliability, vol. 52, no. 1, 2003, pp. 69–74.
11. P. Zeephongsekul, G. Xia, and S. Kumar, "Software Reliability Growth Model: Primary Failures Generate Secondary Faults under Imperfect Debugging," IEEE Trans. Reliability, vol. 43, no. 3, 1994, pp. 408–413.
12. C. Stringfellow, "An Integrated Method for Improving Testing Effectiveness and Efficiency," PhD dissertation, Colorado State Univ., 2000.
13. X. Li et al., "Reliability Analysis and Optimal Version Updating for Open Source Software," Information and Software Technology, vol. 53, no. 9, 2011, pp. 929–936.
14. N. Ullah and M. Morisio, "An Empirical Study of Reliability Growth of Open versus Closed Source Software through Software Reliability Growth Models," Proc. 19th Asia-Pacific Software Eng. Conf. (APSEC 12), 2012, pp. 356–361.
15. K. Sharma et al., "Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach," IEEE Trans. Reliability, vol. 59, no. 2, 2010, pp. 266–276.