Detection of Unexpected Situations by Applying Software Reliability Growth Models to Test Phases

Kiyoshi Honda*, Hironori Washizaki*, Yoshiaki Fukazawa*,
Kazuki Munakata†, Sumie Morita†, Tadahiro Uehara†, and Rieko Yamamoto†
*Waseda University, 3-4-1 Ohkubo, Shijuku-ku Tokyo, JAPAN
Email: khonda@ruri.waseda.jp, {washizaki, fukazawa}@waseda.jp
†Fujitsu Labs Ltd., 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, JAPAN
Email: {munakata.kazuki, morita.sumie, uehara.tadahiro, r.yamamoto} @jp.fujitsu.com

***Abstract*—In software development, software reliability growth models (SRGMs) often provide values that do not meet expectations; sometimes the results of the SRGM and the actual data disagree and other times the SRGM overestimates the expected values. The former often occurs in model curves and the predicted number of faults. For example, the software reliability growth curve cannot describe the situation where developers stop testing multiple times because the equations in SRGMs cannot treat such information. The latter can arise when the total number of expected faults is 100, but the SRGM indicates 1000. If developers encounter such situations, they often doubt the SRGM results and hesitate using SRGMs for predictions. In this study, we apply two different cases of SRGM. Two projects of Fujitsu Labs Ltd. are analyzed using SRGM either for the entire dataset or each test phase. Based on the results and interviews with the developers, we found that the model using separate test phases provides a better fit because faults counted in each test phase have different viewpoints and the deviation between SRGM and expectations indicates a problem with development.**

I. Introduction

Software reliability is an important component of software. To evaluate software reliability, several companies have employed software reliability growth models (SRGMs) in the past few decades [1] [2] [3] [4] [5]. However, SRGMs have several issues. One is that SRGMs sometimes misfit the actual data when development is ongoing. Another is that the results do not always match the developers' expectations.

Herein we apply SRGMs to the datasets of two projects developed by Fujitsu Labs Ltd. in order to determine when SRGMs provide ill-fitted or unexpected results. We assume that the detected faults differ by test phase, and this difference is the source of misfit between the model and actual data. To investigate the source of unexpected results, two different SRGMs are used. In the first case, SRGM is applied to the entire dataset. In the second case, the dataset is divided into test phases, SRGM is applied to each phase separately, and the results are summed. Separating the faults into test phases and combining the results provides a better fitting model.

A. Motivating example

We found two problems when applying SRGMs to an actual dataset. One is an ill-fit between the model and the dataset. The other is when we applied SRGM to a dataset during the middle of development, the values were overestimated compared to the anticipated ones.

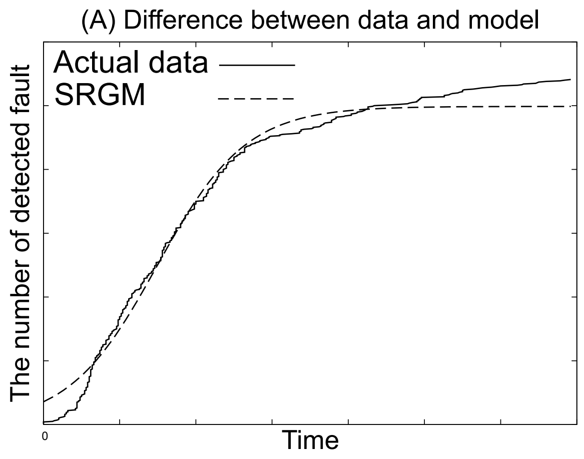

Fig. 1. (A) Difference between the actual data and the model. Solid and dashed lines represent the actual data and SRGM, respectively. Cumulative number of detected faults for all of Project 1 as a function of elapsed time.

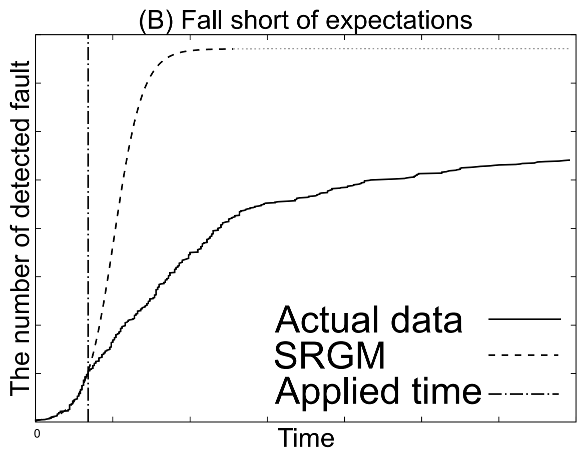

Fig. 2. (B) Case where SRGM overestimates expectations. Solid, dashed, and dotted-dashed lines represent the actual data, SRGM, and the time we applied SRGM, respectively. Cumulative number of detected faults for all of Project 1 as a function of elapsed time.

Figure 1 indicates that the model and the actual data do not fit well during the early, middle, and end of development for the cumulative project. If the model does not fit the actual data, developers and managers cannot decide development plans and release times. Moreover, the misfit leads to difficulties determining when testing is complete. Herein we assume that the misfit is due to the difference between test phases. Test processes are separated by purpose (e.g., unit test, integration test, system test). Therefore, the detected faults depend on the test phase and related modules or features.

Figure 2 indicates that when we applied SRGM to the middle of development (at the dashed-dotted line) the model overestimates the actual data. This means that developers and managers will not believe the model because it indicates that too many faults will be found. It is assumed that when developers and managers apply SRGM depends on the development situation.

This study aims to answer the following research questions:

1) RQ1: How precise is the SRGM model when faults are separated by test phase?
2) RQ2: Can continuous fault predictions and monitoring detect unexpected situations?

II. Background

Software reliability is important when releasing software. Several approaches have been proposed to measure reliability. One is to model fault growth, which is a type of SRGM. Because software development includes numerous uncertainties and dynamics regarding development processes and circumstances, this section explains SRGM, its uncertainties, and dynamics as well as provides a motivating example.

We have proposed a model called the Generalized Software Reliability Model (GSRM) to treat the uncertainties and dynamics regarding development processes and circumstances [6] and studies about predicting release time. We have proposed a method to predict the release time of open source software (OSS) by using GSRM [7] and agile development [8]. Additionally, we have implemented applying GSRM to company's datasets [9].

A. Software Reliability Growth Model (SRGM)

Although many software reliability models have been proposed, the most popular is the non-homogeneous Poisson process (NHPP) model. However, a recent study has suggested the Logistic model is the best model followed by the Gompertz model with regard to fitness [10]. In our study, we employ the Logistic model and Gompertz model using development data containing the number of faults detected for a given time. These models are common in Japan.

The equation of the Logistic model is given by

$$N_{\mathrm{L}}(t) = \frac{N_{\max}}{1 + \exp\{-A_{\mathrm{L}}(t - B_{\mathrm{L}})\}} \tag{1}$$

where $N_{\mathrm{L}}(t)$ is the number of faults detected by time t. If $t \to \infty$, $N_{\mathrm{L}}(t)$ becomes $N_{\max}$. The parameters, $N_{\max}$, A_{L} and B_{L} can be calculated using R [11], which is a language and environment for statistical computing and graphics. The equation of the Gompertz model is given as

$$N_{\mathrm{G}}(t) = N_{\max} \exp(-A_{\mathrm{G}} {B_{\mathrm{G}}}^{t}) \tag{2}$$

where $N_{\mathrm{G}}(t)$ is the number of faults detected by time t. If $t \to \infty$, $N_{\mathrm{G}}(t)$ becomes $N_{\max}$. The parameters, $N_{\max}$, A_{G} and B_{G} can be calculated using R.

B. Project monitoring

Although several methods exist to monitor projects, there are several concerns in software development. The Engineering Project Management using the Engineering Cockpit is one method to manage and monitor project situations [12]. It provides developers and managers with the project specific information.

Nakai et al. studied how to identify the state of a project and the quality of a project based on GQM [13] and project monitoring [14]. They employed Jenkins, which is a continuous integration tool to visualize and collect fault data, lines of codes, test coverage, etc. They tried to judge the status of the project from the collected data based on the GQM method.

Ohira et al. developed the Empirical Project Monitor (EPM), which automatically collects and analyzes data that are versioning histories, mail archives, and issue tracking records from multiple software repositories [15]. EPM provides graphs of collected and analyzed data to help developers and managers. However, EPM is not applicable to analyze SRGMs or to visualize the results.

III. Proposal to detect unexpected situations

We propose the following method to detect unexpected situations using software reliability growth models:

1) Separate faults into the phases that they are detected.
2) Apply SRGM daily to each fault.
3) Detect any unusual situations regarding the predicted number of faults.

The first step clarifies the fault data because the faults detected depend on the phase. For example, faults detected in a unit test relate to a specific module or feature, whereas faults detected in an integration test relate two modules or features. Consequently, the fault level depends on the phase that the fault is detected. Additionally, phases progress differently.

The second step applies the SRGM to the faults by phase to provide the detailed situation of each phase. Moreover, to monitor the behavior of SRGM, we apply it to each fault individually. In this paper, we apply SRGM daily to the data of two projects. We focused on the behavior about the predicted total numbers of faults, which is the model's parameter $N_{\max}$.

The third step monitors the behavior of the SRGM to detect unusual situations. In the motivating example, we mentioned that SRGM sometimes overestimates the expected results. It is assumed that unexpected situations occur in developments. In this paper, we assess when SRGM behaves unexpectedly in developments.

IV. Evaluation and Results

We show the results for two projects on large-scale embedded software developed by Fujitsu Ltd. Herein these projects are identified as Project 1 and Project 2. These projects' qualities have been guaranteed by quality assurance divisions and sufficiently tested. Figures 3 and 4 show the number of faults separated by the test phase when there are eight phases. Although the actual data of Project 1 and Project 2 contain more than eight phases, we treated only eight because the other phases do not have enough faults to model by SRGM.

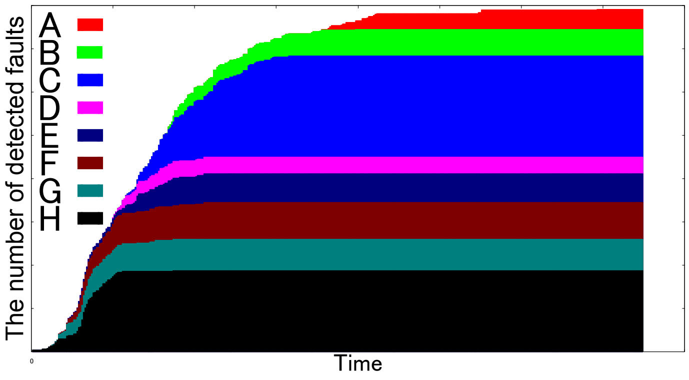

Fig. 3. Cumulative number of detected faults for all of Project 1 represented as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

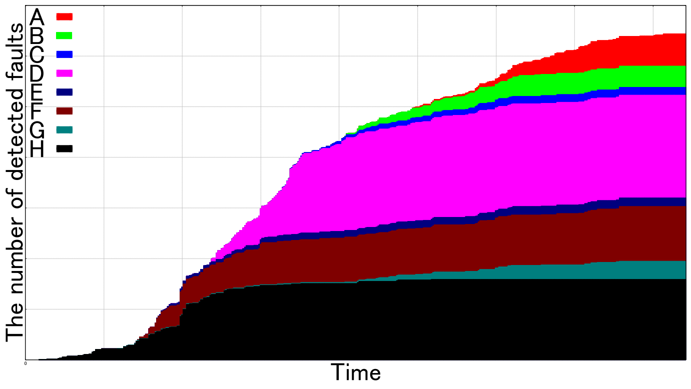

Fig. 4. Cumulative number of detected faults for all of Project 2 represented as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

A. Fitness of model (RQ 1)

We evaluated the fitness of SRGMs in two cases. Case 1 applies SRGM to all the faults in the model simultaneously. Case 2 separates the faults the into eight test phases, applies SRGM to each phase, and then sums the results to treat as one model.

We show the results of Project 1 and 2 in Figs. 5 and 6, respectively. It is should be noted that these figures do not indicate actual values because the information is confidential. The separated faults model (case 2) provides a better fitness than the simultaneous model (case 1). Table I shows the residual sum of squares (RSS) ratio for each model.

These values are divided by the RSS value of case 1. Therefore, the separated model (case 2) for both projects indicates a

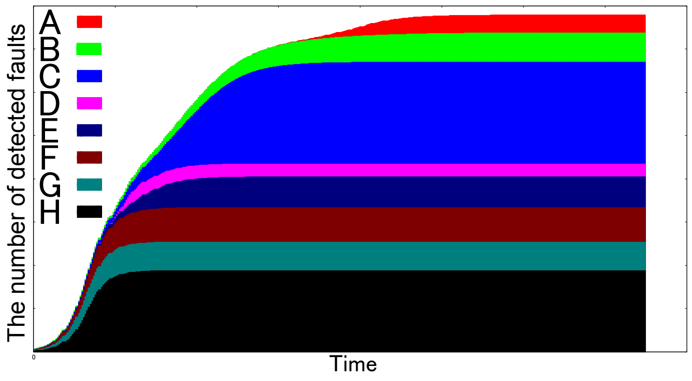

Fig. 5. Cumulative number of predicted faults by SRGMs for all of Project 1 represented as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

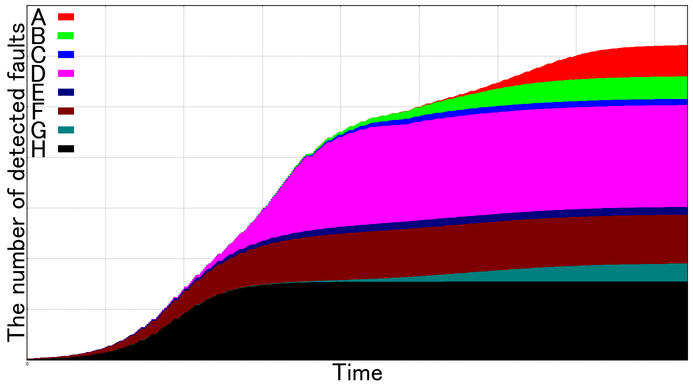

Fig. 6. Cumulative number of predicted faults by SRGMs for all of Project 2 as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

good fitness because the RSS values of the separated models (case 2) are smaller than those of the combined models (case 1).

B. Monitoring Predicted Faults (RQ 2)

We monitored the results of SRGMs by applying them daily to detect unexpected values. Figures 7 and 8 show the results for monitoring the maximum predicted number of faults for Project 1 and Project 2, respectively. Figure 7 has two irregular points when the maximum predicted number of faults is too large, whereas Fig. 8 has five irregular points when the maximum predicted number of faults is too large.

We interviewed the project manager about the situations when the graph indicates an irregular point. In figure 7, the first irregular point coincides with the time that developers thought it was difficult to continue on schedule because several problems remained. The second irregular point is when

TABLE I. COMPARISON OF THE SIMULTANEOUS MODEL (CASE 1) WITH THE SEPARATED MODEL (CASE 2) USING RSS RATIO DATASETS.

	Case 1	Case 2
Project 1	1.000	0.345
Project 2	1.000	0.190

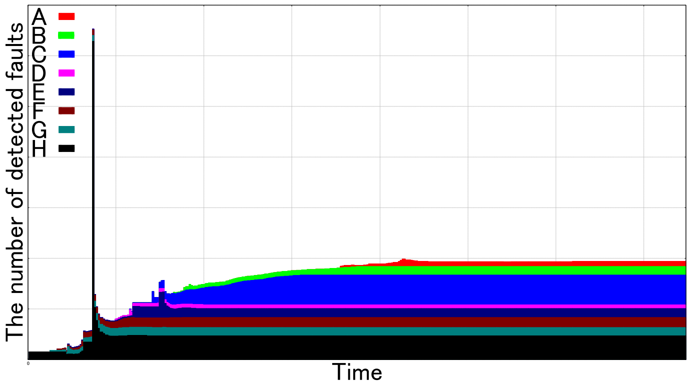

Fig. 7. Cumulative maximum predicted number of faults by SRGMs for all of Project 1 as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

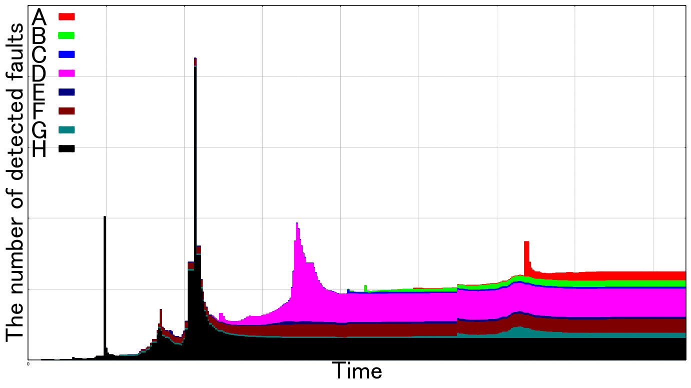

Fig. 8. Cumulative maximum predicted number of faults by SRGMs for all of Project 2 as a function of elapsed time. In the legend, A, B, C, D, E, F, G and H represent the number of faults separated by test phase.

developers tried to reschedule the release plan because several problems reoccurred.

In figure 8, from the first irregular point to the third, several problems occurred intermittently. At the fourth irregular point, several problems reoccurred and developers stopped tests and restarted other tests. At the fifth irregular point, developers detected more faults than ever because they refined and created new test cases.

In the two projects, the irregular points are the same as the unexpected situations. Hence, the results show that monitoring the behavior should detect unexpected situations early.

C. *Thread to validity*

In this paper, we only treat two similar projects from the same organization. However, other researches have treated similar projects in the same organization. Additionally, our results found unexpected situations in the two projects, which we assumed are coincidental and are not related to this research.

V. Conclusion

We successfully obtained a good fitness model by separating faults by test phase and applying SRGM. Moreover, we found unexpected situations in development by monitoring the faults and the behavior of the SRGM. These results demonstrate that if developers and managers monitor the behavior of the SRGM results from the beginning of development, they can detect several unexpected situations earlier than ever.

To provide insight to developers and managers who have trouble with development, we plan to evaluate our method by applying it to ongoing projects and other datasets belonging to other domains or organizations.

References

[1] A. Goel, “Software reliability models: Assumptions, limitations, and applicability,” *Software Engineering, IEEE Transactions on*, vol. SE-11, no. 12, pp. 1411–1423, Dec 1985.

[2] S. Yamada, M. Ohba, and S. Osaki, “S-shaped reliability growth modeling for software error detection,” *Reliability, IEEE Transactions on*, vol. R-32, no. 5, pp. 475–484, Dec 1983.

[3] S. Yamada, M. Kimura, H. Tanaka, and S. Osaki, “Software reliability measurement and assessment with stochastic differential equations,” *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 77, no. 1, pp. 109–116, 1994.

[4] S. Yamada, “Recent developments in software reliability modeling and its applications,” in *Stochastic Reliability and Maintenance Modeling*. Springer, 2013, pp. 251–284.

[5] X. Cai and M. Lyu, “Software reliability modeling with test coverage: Experimentation and measurement with a fault-tolerant software project,” in *Software Reliability, 2007. ISSRE ’07. The 18th IEEE International Symposium on*, Nov 2007, pp. 17–26.

[6] K. Honda, H. Washizaki, and Y. Fukazawa, “A generalized software reliability model considering uncertainty and dynamics in development,” in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science, J. Heidrich, M. Oivo, A. Jedlitschka, and M. Baldassarre, Eds. Springer Berlin Heidelberg, 2013, vol. 7983, pp. 342–346.

[7] K. Honda, H. Washizaki, and Fukazawa, “Predicting release time based on generalized software reliability model (gsrm),” in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*. IEEE, 2014, pp. 604–605.

[8] H. Washizaki, K. Honda, and Fukazawa, “Predicting release time for open source software based on the generalized software reliability model,” in *Agile Conference (AGILE), 2015*, August 2015.

[9] K. Honda, H. Nakai, H. Washizaki, Y. Fukazawa, K. Asoh, K. Takahashi, K. Ogawa, M. Mori, T. Hino, Y. HAYAKAWA *et al.*, “Predicting time range of development based on generalized software reliability model,” in *21st Asia-Pacific Software Engineering Conference (APSEC 2014)*, 2014.

[10] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Torner, “Evaluating long-term predictive power of standard reliability growth models on automotive systems,” in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, Nov 2013, pp. 228–237.

[11] “The r project for statistical computing,” `http://www.r-project.org/`.

[12] T. Moser, R. Mordinyi, D. Winkler, and S. Biffl, “Engineering project management using the engineering cockpit: A collaboration platform for project managers and engineers,” in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, July 2011, pp. 579–584.

[13] V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach,” in *Encyclopedia of Software Engineering*. Wiley, 1994.

[14] H. Nakai, K. Honda, H. Washizaki, Y. Fukazawa, K. Asoh, K. Takahashi, K. Ogawa, M. Mori, T. Hino, Y. Hayakawa, Y. Tanaka, S. Yamada, and D. Miyazaki, “Initial industrial experience of gqm-based product-focused project monitoring with trend patterns,” in *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, vol. 2, Dec 2014, pp. 43–46.

[15] M. Ohira, R. Yokomori, M. Sakai, K.-i. Matsumoto, K. Inoue, and K. Torii, “Empirical project monitor: A tool for mining multiple project data,” in *International Workshop on Mining Software Repositories (MSR2004)*. IET, 2004, pp. 42–46.