SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2017; **00**:1–23
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

Debugging-Workflow-aware Software Reliability Growth Analysis

Marcello Cinque, Domenico Cotroneo, Antonio Pecchia,
Roberto Pietrantuono, Stefano Russo

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II
Via Claudio 21, 80125 Naples, Italy.

SUMMARY

Software reliability growth models (SRGMs) support the prediction/assessment of product quality, release time and testing/debugging cost. Several SRGM extensions take into account the bug correction process. However, their estimates may be significantly inaccurate when debugging fails to fully fit modeling assumptions.
This paper proposes DWA-SRGM, a method for reliability growth analysis leveraging the debugging data usually managed by companies in bug tracking systems. Based on a characterization of the debugging workflow within the software project under consideration (in terms of bug features and treatment phases), DWA-SRGM pinpoints the factors impacting the estimates, and to spot bottlenecks, thus supporting process improvement decisions. Two industrial case studies are presented, a Customer Relationship Management system and an Enterprise Resource Planning system, whose defects span a period of about 17 and 13 months, respectively. DWA-SRGM revealed effective to obtain more realistic estimates, and to capitalize on the awareness of critical factors for improving debugging.

Received . . .

KEY WORDS: Software reliability, Reliability models, SRGM, Debugging, Release planning

1. INTRODUCTION

1.1. Motivation

Testing and debugging are essential to improve software reliability, as they allow to detect and correct faults before product release. The analysis of the reliability level as testing and debugging proceed has been the subject of a large body of research work. A common approach makes use of *software reliability growth models* (SRGMs), a wide class of models to fit inter-failure times from testing data. SRGMs serve for many purposes [1, 2, 3]: *i)* to estimate the number of residual faults; *ii)* to predict reliability given the testing budget; *iii)* to schedule release time with a target reliability level; *iv)* to compare actual and estimated release times, so as to identify delays and their causes.

SRGMs are usually based on several assumptions [4]. Significant attention has been devoted to the one of *immediate debugging* - i.e., that corrections take zero time - which hardly holds in real projects. SRGM extensions have been proposed to this aim [5, 6, 7]; nonetheless, empirical studies have made it clear that debugging is a complex process [8, 9, 10], hardly modeled analytically.

*Correspondence to: Stefano Russo, DIETI - Università degli Studi di Napoli Federico II, Email: sterusso@unina.it.

The workflow of debugging is influenced by many factors, including: bug[†] type, severity, priority, policies for assignment to debuggers, skills of debuggers, release policy. They may result into high and strongly variable fault correction times, which undermine the seamless incorporation of debugging in a SRGM. For large software projects, this may mislead SRGM-based decisions, causing for instance the early release of a version with too many residual defects.

Nowadays, the use of open source or commercial *defect management systems* (DMSs) such as Bugzilla [11] and JIRA [12] - also called *issue trackers* or *bug tracking systems* - is widely spread, with tens of thousands of installations worldwide. Bug data (type, priority, assignment, severity, etc.) are thus available to the organizations, but in our experience they are rarely used in conjunction with SRGMs to drive key decisions such as product reliability estimate and release schedule.

In real world software development projects it is important to assess the actual impact of debugging on reliability estimates and product release decisions during advanced testing phases. To this aim, field data from DMSs can be leveraged to tailor SRGM models. Moreover, they make it possible to spot bottlenecks in the debugging activities, enabling process improvement decisions.

1.2. Contribution

This paper proposes DWA-SRGM (Debugging-Workflow-Aware SRGM), a method for reliability growth analysis driven by defect data available from issue trackers. It allows to assess the impact of non-ideal debugging on SRGM-based inferences, such as *expected quality*, *release time*, and *marginal cost/benefit ratio* of testing and correction. DWA-SRGM builds an empirical characterization of the organization's debugging process based on a variety of factors - namely, the severity of bugs, the functionality impacted, the duration of treatment activities the defects go through. This allows to quantify the extent to which debugging impacts SRGM-based estimates, and in turn the consequent decisions. Software process managers can exploit the information about the most-impacting debugging factors as feedback to improve the debugging process.

DWA-SRGM is meant to be applied in advanced testing phases, from integration test to system test, when decisions on release times become relevant. It is not meant to be used since unit test, when the product is not integrated yet and still far from release. The method leads to several practical implications and recommendations:

- Assuming immediate debugging may cause significant overestimation of the product quality at release time. This may lead to erroneously anticipate the product release, with impact on maintenance costs, customer satisfaction and company reputation.
- Cost is a primary industry driver: SRGMs under the immediate debugging assumption may cause underestimation of the actual cost.
- The analysis of issue tracking data is highly valuable to characterize the actual debugging process. SRGM-based inferences can be much improved using this characterization, allowing to spot bug features and debugging factors mostly impacting quality. Ultimately, this is extremely useful for process improvement decisions, including identifying bottlenecks and corrective actions in bug treatment activities.

Two real-world software projects are described as case studies, namely a Customer Relationship Management (CRM) system and an Enterprise Resource Planning (ERP) system, whose debug data have been made available in the ICEBERG project[‡] by an industrial partner. The impact of debugging on quality, release time and cost estimate is quantified; for instance, it is shown that, under the immediate debugging assumption, releasing the CRM at 75% of the estimated number of defects would overestimate its quality by 16% and underestimate the release time by 26% – potentially causing remarkably wrong decisions. Moreover, it is shown how to spot debugging bottlenecks and product functionalities mostly influencing estimates/predictions.

[†]The literature on debugging typically uses the terms *defect* or *bug*, while in the SRGM field the term *fault* is used: in this work they are used as synonyms to denote the adjudged cause of an observed failure.
[‡]ICEBERG is an industry-academia transfer of knowledge project funded by EU, focused on the estimation of software quality and its impact on costs (www.iceberg-sqa.eu).

DOI: 10.1002/stvr

In the following, the related work is surveyed in Section 2. An overview of the DWA-SRGM method is given in Section 3. The two case studies are introduced in Section 4. The first stage of DWA-SRGM is described in Sections 5 and 6. The second stage is shown in action in Section 7. Section 8 discusses findings and limitations. Section 9 contains concluding remarks.

2. RELATED WORK

The literature on software reliability modeling (SRM) is huge. Febrero *et al.* have performed a Systematic Literature Review of SRM [13], pointing out that software reliability has multiple definitions, from continuity of correct service, to the standard-based view of reliability as a concept which has to consider several stakeholders' viewpoints, and combines availability, maturity, fault tolerance and recoverability. DWA-SRGM assumes for reliability the probability of correct operation in the time interval $(0, t)$. Moreover, Febrero *et al.* state that if reliability models "*are to be effectively applied ... they have to be clearly valuable and profitable for the organization*". DWA-SRGM shares this view, addressing the specific problem of reliability analysis driven by debugging data available in an organization. Hence, the following analysis of the related work focuses on debug-aware models and on empirical characterization of debugging.

2.1. Debugging-aware reliability modeling

Software reliability models aim to predict or to assess the reliability of a software product [14, 15]. There is a wide variety of models, among which SRGMs are the most successful ones; they are typically applied during testing. Many SRGMs have been proposed to capture several facets of the fault detection process. Those based on non-homogeneous Poisson processes (NHPP) are particularly popular. A seminal model was proposed by Goel and Okumoto in 1979 [16]: it describes the fault detection process by an exponential *mean value function* distribution. Other models include: the generalized version of the exponential model, using the Weibull distribution [17]; the S-Shaped model [18], conceived to capture the possible increase/decrease of the failure rate during testing; the Gokhale and Trivedi log-logistic model [19], which also follows an increasing/decreasing pattern describing the initial phase of testing as characterized by a slow learning phase. More recent models, as those based on the Gompertz SRGM proposed by Ohishi *et al.* [20], derive from the statistical theory of extreme-value. Several tools are available to deal with model parameterization and fitting (e.g., SoRel [21], PISRAT [22], and CASRE [23]). Besides for planning tests of the system as a whole, these models and tools are used to optimize the testing effort distribution to system components, i.e. for solving *testing effort allocation problems* [24, 25, 26, 27].

Most NHPP-based SRGMs share a set of assumptions. These include: perfect debugging, immediate debugging, dependent inter-failure times, no duplicate defect reports, no code changes during testing, equal probability to find a failure over time [4]. Some models work well even when some assumptions are partially violated [28]. However, as for immediate debugging, the time to repair defects appears to be less and less negligible as size and complexity of systems grow.

Initial work modeling fault removal dates back to the '70s. The separate modeling of detection and correction was proposed by Schneidweind [29]. He modeled debugging as a process following fault detection with a constant time delay. Later, the model was extended considering more realistic time-dependent models (Xie and Zhao [30] in 1992, Schneidweind [31] in 2001). Non-homogenous Markov chains were also proposed to model these two processes in a unified way [32, 33]. The incorporation of the debuggers' *learning effect* into SRGM modeling has been explored by Ho *et al.* [34] with stochastic differential equations for the correction process. Queuing models for non-immediate fault corrections have been proposed by Musa et al. [35] and by Huang and Huang [7].

More recently, general frameworks have been proposed, where NHPP-based fault detection models are adjusted to consider several forms of the time-dependent fault correction. Examples are those proposed by Lo and Huang [5] (used in DWA-SRGM), where the fault correction mean value function is derived from the fault detection one, and by Kapur *et al.* [6], who consider non-immediate and imperfect debugging. Wu *et al.* [36] study the joining of an exponential fault detection model with exponential, normal, and gamma delay distribution models for fault correction.

DOI: 10.1002/stvr

2.2. *Empirical characterization of the debugging process*

SRGMs model correction time distributions regardless of the various debugging activities and of the factors influencing their duration. These have been investigated recently by several authors along a separate research trend on the empirical characterization of debugging. Nguyen *et al.* presented an analysis of 1,500 defects from bug reports in 5 years for an IBM middleware, showing that the time to repair is influenced by developer expertise and by the topic [8]. Zhang *et al.* studied three open source software and found the following factors influencing the time lag between the defect assignment to a developer and the actual starting of the repair action: the assigned severity; the bug description; the number of methods and code changes [9]. Ihara *et al.* focused on bottlenecks in the bug management process of the Apache web-server and the Firefox browser [10]; the main cause of inefficiency was the time lag between the repair and its verification confirming the correctness of the resolution. Bug fixing was analyzed in an industrial context by Carrozza et al., highlighting a remarkable heterogeneity among components in large-scale critical systems [37].

The debugging delays induced by all these factors can make SRGM-based estimates inaccurate. In a previous work, fixing times were shown to impact the accuracy of the quality estimation made during testing [38]. Moving from that evidence, DWA-SRGM introduces a characterization of the debug steps for *debugging-workflow-aware reliability growth modeling*, allowing to analyze factors impacting quality prediction and release schedule. This approach complements that of Peng *et al.* [39], who considered the influence of testing effort on the debugging delay, and dealt with imperfect debugging, i.e. with modeling the introduction of new faults when correcting a detected fault.

3. METHOD OVERVIEW

DWA-SRGM acts in two stages (Figure 1). The first one is based on the mathematical modeling framework described in Section 5.1. Initially, the SRGM which best fits defects *detection* data (Section 5.2) is selected. Then, the SRGM is extended with the correction times distribution, yielding a *correction-aware SRGM* (Section 5.3). This is used for reliability or cost estimates and for release time prediction. The results for the CRM and ERP systems are presented in Section 6.

The second stage starts with the characterization of the debugging workflow. Organizations usually set up personalized installations of *issue tracking systems*, depending on their process management choices. Testing reveals the presence of faults, which then undergo a correction workflow that can well be reconstructed from tracked data. These include the typical attributes of fault *severity*, treatment *state* (open, assigned, ..., closed), state transition *timestamp*, etc. Treatment states are grouped to identify major workflow phases, such as *waiting* to be assigned, *being worked*, *under testing*. The characterization ends with the computation of basic statistics. The extended SRGM (output of the first stage) is then applied to workflow phases, enabling analysis of their impact on the estimation/prediction of the quality at release and of the optimal release time. Similarly, the model is applied to spot the impact of bug features of interest, such as severity or impacted product functionality. Such fine-grained analyses support conscious process improvement decisions. The second stage is described in action for the CRM case study in Section 7.

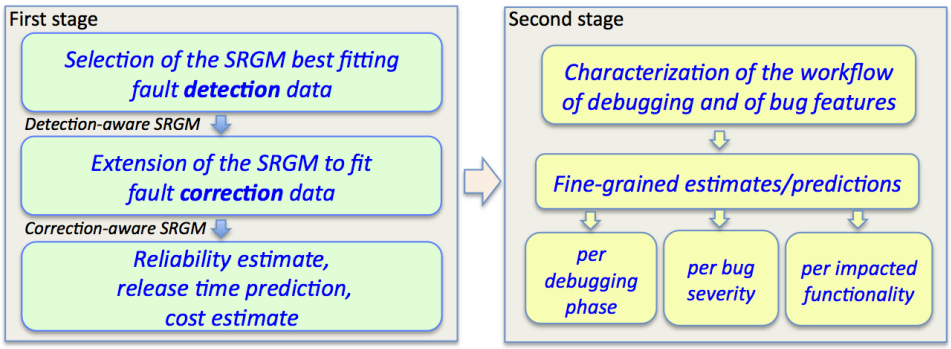


Figure 1. The DWA-SRGM method.

DOI: 10.1002/stvr

4. CASE STUDIES

The datasets are bug reports gathered from the defect management systems of real software projects from two multinational companies[§]. The first concerns a CRM product. It has a classical three-layer architecture, with Front-end, Back-end and Database layers, glued together by an Enterprise Application Integration middleware. It provides typical CRM services: sales management, customer folder, agenda, user profile, inventory, supplying, payments, and reporting. The issues span a period of about 1.5 years, from September 2012 to January 2014, and are related to advanced testing phases, namely integration and system tests (performed by 22 human resources). The removal of a few inconsistencies (e.g., duplicated issues) led to the set of 602 bugs analyzed in this study. CRM bug reports are characterized by the following attributes (samples are listed in Table I):

- *ID*: a unique ID for the issue;
- *Severity*: the severity of the issue, classified in *blocking*, *major* and *minor*;
- *Status*: issues traverse nine working statuses, according to the workflow adopted: *new*, *published*, *in_study*, *launched*, *completed*, *tested*, *delivered*, *suspended*, and *closed*;
- *Created on*: the timestamp of creation;
- *Component*: the name of the software component affected by the issue;
- *Functionality*: the name of the functionality, within the component, affected by the issue;
- *Updated on*: the timestamp of status transition; the main ones are related to opening and closing (issue solved); the timestamps of intermediate transitions are useful to analyze the phases of the debug process, e.g., how long an issue waited to be fixed;
- *Resolution*: final classification of the issue, as *fixed*, *won't_fix*. It is used to indicate defects that cannot be fixed, even if their processing can be declared as closed.

Additional information about the statuses traversed by every issue is noted in a different report, useful to reconstruct the processing history of each bug. More details are provided in Section 7.

Table I. Examples of bug records for the CRM system (date format: DD/MM/YY).

ID	Severity	Status	Created on	Component	Functionality	Updated on	Resolution
CRM-2	blocking	closed	07/09/12 10:16	Front end	Incomes Management	21/09/12 12:02	fixed
CRM-214	minor	closed	01/10/12 10:44	Front end	Customer Folder	06/10/12 15:02	fixed
CRM-273	major	closed	05/10/12 09:56	EAI	Customer	19/10/12 17:47	fixed

The second dataset is an ERP system for a telecommunication company. It is used for a variety of businesses processes, including billing, activation/termination of customer contracts, communication links installation and repair, management of third party suppliers. The dataset consists of 7,610 bugs collected during advanced testing phases from April 2013 to April 2014. The ERP encompasses about 100 functionalities. Table II shows an excerpt of the ERP dataset. Besides a unique ID, each bug report is characterized by the following attributes:

- *Component*: the software module affected by the issue;
- *Status*: issues in this dataset can assume three working statuses: *open*, *closed*, and *rejected*;
- *Severity*: severity of the issue, classified in *blocking*, *very high*, *high*, *medium*, and *low*;
- *Priority*: priority of resolution, classified in *urgent*, *very high*, *high*, *medium*, and *low*;
- *Type*: the type of the issue, indicating, e.g., if the defect is related to functional, integration, or delivery issues;
- *Detected*: the opening timestamp of the issue;
- *Closed*: the closing timestamp of the issue.

Table II. Examples of bug records for the ERP system (date format: DD/MM/YY).

ID	Component	Status	Severity	Priority	Type	Detected	Closed
54506	Component_1	closed	medium	medium	integration	23/04/13	26/04/13
55746	Component_2	closed	blocking	urgent	delivery/setup	11/06/13	11/06/13
59442	Component_3	closed	very high	urgent	functional	25/11/13	09/12/13

[§]The datasets are available upon request for verification purposes. Prospective applicants can contact the corresponding author to obtain instructions on how to access an anonymized version of the datasets, and related terms of usage.

DOI: 10.1002/stvr

5. CORRECTION-AWARE SRGM MODELING

5.1. Modeling framework

The most common subclass of SRGMs, those describing the failing process as a non-homogeneous Poisson process (NHPP), is considered here. They are characterized by the mean value function (*mvf*), $m(t)$, that is the expectation of the cumulative number of defects $N(t)$ detected at time t: $m(t) = E[N(t)]$. The *mvf* provides indications on how testing is proceeding, namely on how many faults are being detected over time, and how many faults are expected to be found at a certain testing time t. The different types of SRGMs can be described by their *mvf*. This can be written as $m(t) = a \cdot F(t)$, where a is the expected number of total faults, and $F(t)$ is a distribution function, that can take several forms depending on the detection process. Hereinafter, the mean value function of a SRGM referring to the fault detection (correction) process is denoted with $m_d(t)$ ($m_c(t)$).

According to Lo and Huang [5] and to Schneidweind [31], the mean number of faults detected in the time interval $(t, t + \Delta t]$ is assumed proportional to the mean number of residual faults; similarly, the mean number of faults corrected in $(t, t + \Delta t]$ is assumed to be proportional to the mean number of detected but not yet corrected faults. This proportionality is expressed by the fault detection and fault correction rates per fault as functions of time, denoted with $\lambda(t)$ and $\mu(t)$. From this, the mean value function of the fault detection and correction processes are expressed respectively as:

$$\frac{dm_d(t)}{dt} = \lambda(t)(\alpha - m_d(t)), \quad \alpha > 0 \tag{1}$$

$$\frac{dm_c(t)}{dt} = \mu(t)(m_d(t) - m_c(t)) \tag{2}$$

where α is the estimated initial number of faults. In case of constant fault detection rate ($\lambda(t) = \beta$), $m_d(t)$ is the Goel-Okumoto exponential SRGM ($m_d(t) = \alpha(1 - e^{-\beta t})$).

The function $m_c(t)$ needs to be modeled realistically, other than assuming, as it is often done, a constant correction time Δ – where $m_c(t) = m(t - \Delta)$. It can be shown that using Equations 1 and 2 to describe the fault detection and correction processes, and defining $D(t) = \int_0^t \lambda(s)ds$, and $C(t) = \int_0^t \mu(s)ds$ (i.e., the cumulative detection and correction rate per fault, respectively), the cumulative number of detected and corrected faults are given by [5]:

$$m_d(t) = \alpha(1 - e^{-D(t)}) \tag{3}$$

$$m_c(t) = e^{-C(t)} \left(\int_0^t c(s)e^{C(s)} m_d(s)ds \right) = e^{-C(t)} \left[\int_0^t \alpha c(s)e^{C(s)} \left(1 - e^{-D(s)}\right) ds \right]. \tag{4}$$

DWA-SRGM uses this modeling framework for the bug detection and correction processes. For instance, if the detection process follows an exponential SRGM with parameter β ($\lambda(t) = \beta$), and the correction time is exponentially distributed with parameter γ ($\mu(t) = \gamma$), then:

$$m_c(t) = \alpha \left(1 + \frac{\gamma}{\beta - \gamma} e^{-\beta t} - \frac{\beta}{\beta - \gamma} e^{-\gamma t} \right). \tag{5}$$

5.2. Step 1: Selection of the SRGM best fitting the fault detection data

DWA-SRGM starts determining the SRGM best fitting the actual fault detection times from the SRGMs listed in Table III, chosen because of their wide spread and of their ability to capture several potential behaviors of the testing process. The list includes: the model by Goel and Okumoto [16], one of the most successful and popular; the Delayed S-Shaped [18] and log-logistic curves [19], suited for possible behaviors where the initial slow increase represents the gradual improvement of testers' skills, and where faults are mutually dependent (i.e., some faults are detectable only after some others); the log-normal SRGM, which demonstrated a noticeable ability to fit a wide variety of scenarios and failure data collected in real projects [40, 41].

Data are fitted by the EM algorithm [42] to estimate parameters for each SRGM. In both case studies, the SRGMs with the lowest value of the *Akaike Information Criterion* (AIC) is selected. Figure 2 shows the number of faults detected and the fitting SRGM for the studied systems.

DOI: 10.1002/stvr

Table III. Software Reliability Growth Models.

Model	*m(t) function*		*Model*	*m(t) function*
Exponential	$\alpha \cdot (1 - e^{-\beta t})$		Log-Logistic	$\alpha \cdot \frac{(\lambda t)^{\kappa}}{1+(\lambda t)^{\kappa}}$
S-shaped	$\alpha \cdot [1 - (1 + \gamma t)e^{-\gamma t}]$		Log-Normal	$\alpha \cdot \Phi(\frac{log(t)-\mu}{\sigma})$ Φ is the normal distribution

Table IV shows the statistics of the selected models, which turned out to be the exponential one for both cases. Both models have a high coefficient of determination ($R^2 = 0.98$ and $R^2 = 0.99$) and low values of the percentage *Relative Prediction Error* (*%RPE*), meaning high prediction ability. Specifically, the *%RPE* at time t_k is computed as: $\%RPE = \frac{(m(t_k)-y_k)}{y_k} \cdot 100$, where $m(t_k)$ is the predicted number of detected faults (or corrected faults, in the following cases of correction-aware SRGMs) at time t_k and y_k is the actual number of detected (or corrected) faults at the same time [43]. The models were trained with data collected on a time window equal to the 2/3 of the total observation time (e.g., 210 days out of 315 for the CRM dataset, 244 days out of 365 for the ERP dataset), and the %RPE computed on the latest observation time (e.g., $t_k = 315$ and $t_k = 365$ for CRM and ERP datasets, respectively). The results show that in the case of the CRM, where the testing process is at an advanced stage, the *%RPE* is very low (0.25%); in the ERP case, which is at an earlier stage of testing, the *%RPE* is still able to provide an error under 4%.

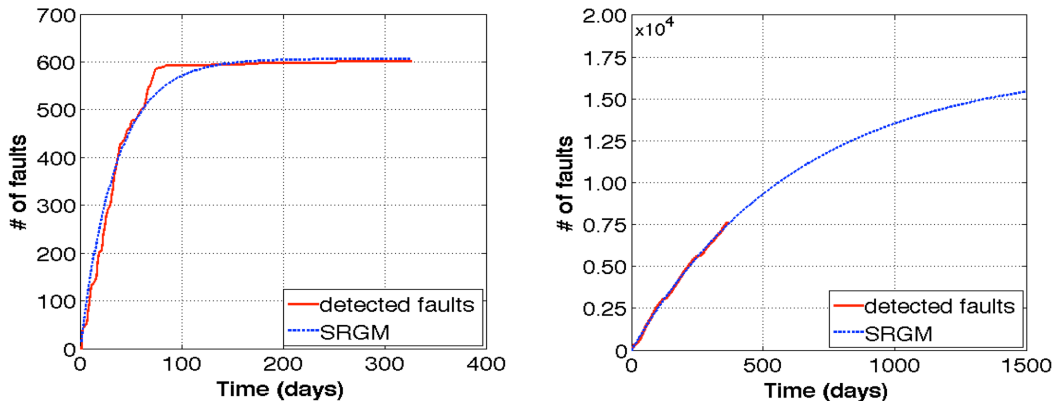
Table IV. Results of SRGM fitting for the CRM and ERP systems

Dataset	*#Tracked Faults*	*Selected SRGM*	*Exp. #Faults at $t = \infty$*	*Rate param.*	*AIC*	R^2	*%RPE*
CRM	602	Exponential	603.04	3.04E-02	-1.13E03	0.98	-0.25%
ERP	7,610	Exponential	16,571.23	-1.68E-03	-31121.93	0.99	-3.44%

The models provide estimates of: residual faults at a given time; percentage of detected faults over the total expected ones; fault detection rate; expected reliability. Note that these metrics are mutually equivalent: the expected cumulative number of faults at time t is the mvf, whose first derivative is the failure intensity function $\lambda(t)$; the latter can be used in the computation of reliability [19, 25].

For the CRM dataset the process detected more than 99.8% of total expected faults, and it is evident that the testing process is close to saturation, detecting less and less faults as it proceeds. Figure 2 suggests, intuitively, that an earlier release could be more advantageous from the cost point of view, as after day 100 further faults are found only after a long time. The model is meant to support a quantitative assessment of this type of remarks.

This analysis refers to detected faults, i.e., a "quality" of 99% means that 99% of the total estimated faults have been detected: this is the actual released quality *only under the assumption*



(a) CRM (b) ERP

Figure 2. Cumulative number of opened faults and SRGM for the CRM and ERP systems.

DOI: 10.1002/stvr

that the correction of those faults is immediate. The delivered quality is determined by the *removed* faults, whose correction contributes to the reliability growth.[¶]

5.3. Step 2: Extension of the SRGM fitting fault correction data

The fitting of the correction data is tried with several distributions, adopting the maximum likelihood estimator (MLE) for the parameters of the joint density of detected and corrected fault counts [44][‖]:

$$L = f(n, m, i = 1, \ldots, k|\theta) =$$

$$= \prod_{\substack{i=1,\ldots,k \\ m_i<n_{i-1}}} e^{-[m_d(t_i)-m_d(t_{i-1}|\theta]} \frac{[m_d(t_i) - m_d(t_{i-1})|\theta]^{n_i-n_{i-1}}}{(n_i - n_{i-1})!} \cdot e^{-[m_c(t_i)-m_c(t_{i-1}|\theta]} \frac{[m_c(t_i) - m_c(t_{i-1})|\theta]^{m_i-m_{i-1}}}{(m_i - m_{i-1})!}$$

$$\times \prod_{\substack{i=1,\ldots,k \\ m_i>n_{i-1}}} e^{-[m_d(t_i)-m_d(t_{i-1}|\theta]} \frac{[m_d(t_i) - m_c(t_{i-1})|\theta]^{n_i-m_i}}{(n_i - m_i)!} \frac{[m_c(t_i) - m_d(t_{i-1})|\theta]^{m_i-n_{i-1}}}{(m_i - n_{i-1})!} \tag{6}$$

where: $\theta \in \Theta$ are the parameters of interest in the detection and correction models; n_i and m_i are the number of faults detected and corrected, respectively, at time t_i (m_0=n_0=0). This is a general form of the likelihood function for the combined detection/correction process: it describes the non-homogeneous Poisson process at instant t_i, separated in the two cases (namely, the two product terms) of *i)* number of cumulative corrected faults at time t_i being less than number of cumulative detected faults at the previous time t_{i-1} (i.e., $m_i < n_{i-1}$), and *ii)* the opposite case (i.e., $m_i > n_{i-1}$), in which the Poisson distribution is decomposed in a slightly different form (avoiding to have a number of cumulative corrected faults greater than the number of detected ones). The MLE of parameters is $\hat{\theta}$ =argmax$_{\theta\in\Theta}\hat{L}(\theta)$.

The MLE in Equation 6 is applied considering the exponential fault detection model and the following correction time distributions: *i)* Exponential, which represents well the fixing times [31, 35]; *ii)* Normal, suitable if assuming faults of equal size; *iii)* Gamma, to consider a more general case based on the exponential time delay.[**] Denoting with α and β the parameters of the exponential detection model (α representing the expected number of detected faults, and β the detection rate per fault), the corresponding specialization of Equation 4 for these three models are [36]:

$$i) \qquad m_c(t) = \alpha\left(1 + \frac{\gamma}{\beta-\gamma}e^{-\beta t} - \frac{\beta}{\beta-\gamma}e^{-\gamma t}\right) \tag{7}$$

with γ being the correction rate per fault of the fault correction exponential distribution;

$$ii) \quad m_c(t) = \\ -\alpha e^{-\beta t+\gamma\beta+\beta^2\sigma^2/2}\left[\Phi(t, \beta\sigma^2 + \mu, \sigma) - \Phi(t0\beta\sigma^2 + \mu, \sigma)\right] + \alpha\left[\Phi(t, \mu, \sigma) - \Phi(0, \mu, \sigma)\right] \tag{8}$$

with μ and σ being the mean and standard deviation of the fault correction normal distribution, and Φ the cumulative distribution function of the standard normal distribution;

$$iii) \qquad m_c(t) = \alpha\Gamma(t, \delta, \kappa) - \frac{\alpha e^{-\beta t}}{(1-\beta\kappa)}\Gamma(t, \delta, \frac{\kappa}{(1-\beta\kappa)}) \tag{9}$$

where δ and κ are the parameters of the fault correction Gamma distribution, denoted with Γ.

[¶]Quality is expressed here by the ratio (or percentage) of closed issues over the total number of issues. For what said about the equivalence of this information to failure intensity and reliability, "quality estimation" and "quality prediction" (referring to the current or a future time) are equivalent to "reliability estimation" and "reliability prediction".

[‖]The popular maximum likelihood estimate is adopted here to estimate model parameters. Other approaches are feasible, including the EM algorithm in [42], and the genetic algorithm in [45], and could replace MLE in DWA-SRGM.

[**]Note that to keep the treatment simple, not all combinations of detection and correction models are considered, but only the combinations of correction models with the selected detection model. This two-steps approach yields high accuracy as it considers several detection and correction models; however, it clearly gives a sub-optimal solution.

DOI: 10.1002/stvr

Tables V (for CRM) and VI (for ERP) list the parameter values and the R^2 goodness of fit metric and the *%RPE* prediction accuracy metric for each tried correction model. Gamma and Normal models show the best fitting and prediction ability – hence they are selected as best representatives of the fault correction process in the two cases.

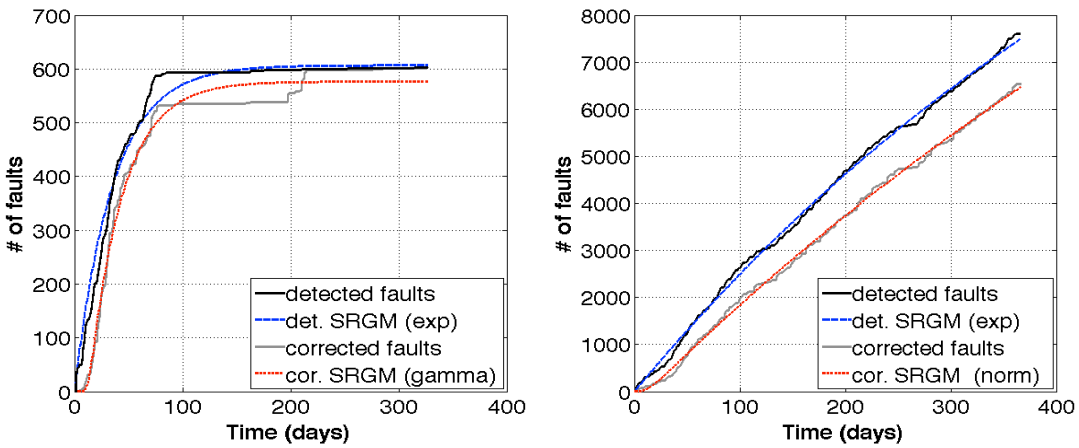
Table V. Models for the fault correction process for the CRM system.

Model	*Parameters*				R^2	*%RPE*
Exp	$\hat{\alpha}$ = 577.03	$\hat{\beta}$ = 0.033	$\hat{\gamma}$ = 0.101	-	0.971	-0.42%
Norm	$\hat{\alpha}$ = 591.61	$\hat{\beta}$ = 0.023	$\hat{\mu}$ = 0.133	$\hat{\sigma}$ = 0.056	0.954	1.75%
Gamma	$\hat{\alpha}$ = 576.54	$\hat{\beta}$ = 0.032	$\hat{\mu}$ = 9.704	$\hat{\kappa}$ = 1.032	0.974	-0.19%

Table VI. Models for the fault correction process for the ERP system.

Model	*Parameters*				R^2	*%RPE*
Exp	$\hat{\alpha}$ = 20,300	$\hat{\beta}$ = 0.001	$\hat{\gamma}$ = 0.078	-	0.9985	0.28%
Norm	$\hat{\alpha}$ = 24,460	$\hat{\beta}$ = 0.001	$\hat{\mu}$ = 9.409	$\hat{\sigma}$ = 10.706	0.9986	-0.23%
Gamma	$\hat{\alpha}$ = 19,600	$\hat{\beta}$ = 0.001	$\hat{\mu}$ = 2.809	$\hat{\kappa}$ = 4.597	0.9986	-0.45%

Figure 3 plots the fault detection and the fault correction curves, along with the raw data about detected and corrected faults. The difference between the two curves represents the error that the modeler commits assuming immediate repair in estimating reliability (in terms of number of remaining faults) for a given release time, or in predicting the release time to achieve a given reliability value.



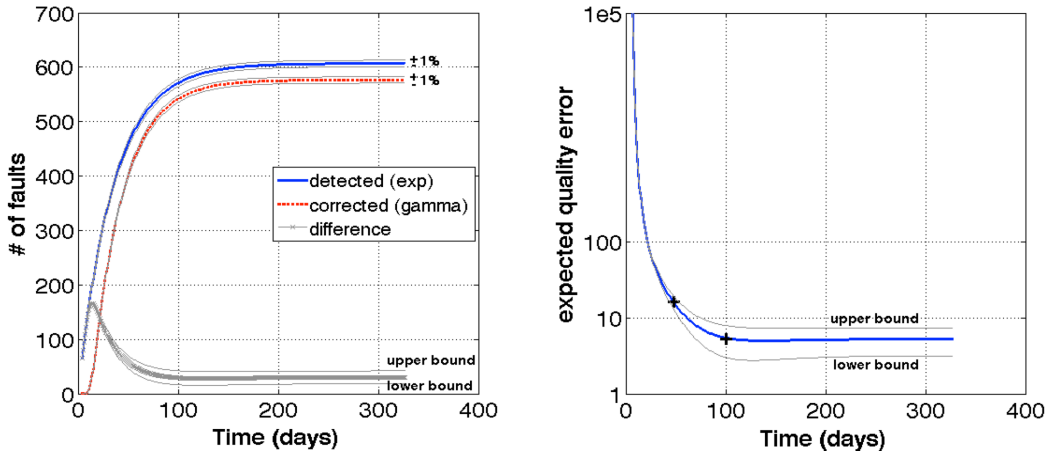
(a) CRM (b) ERP

Figure 3. Fault detection ($m_d(t)$) and correction ($m_c(t)$) *mvf*s for the CRM and ERP systems.

6. IMPACT OF THE DEBUGGING TIME

Debugging introduces latency between the time a fault is detected and the time it is corrected. The CRM and ERP datasets are now analyzed to measure the impact of the **debugging time** on SRGM-based predictions in terms of: *(i)* expected quality at a given release time; *(ii)* release time to achieve a given quality level; *(iii)* cost for testing and debugging depending on the release time. The **sensitivity** of the predictions with respect to variations of modeling parameters is also assessed.

DOI: 10.1002/stvr



(a) Predicted number of detected and corrected faults. (b) Expected quality error (y-axis in logarithmic scale).

Figure 4. Impact on reliability estimate for the CRM system.

6.1. *Impact on reliability estimate*

6.1.1. Metric. Under the immediate debugging assumption, the expected number of corrected faults at a given time is supposed to be equal to the number of detected faults, estimated by $m_d(t)$. By removing this assumption, the expected number of corrected faults is given by $m_c(t)$, which fits the actual correction times. The **percentage error on the expected quality** is given by:

$$\epsilon_Q(t) = \frac{m_d(t) - m_c(t)}{m_c(t)} \cdot 100. \tag{10}$$

Equation 10 quantifies the relative difference in the expected quality *under ideal (immediate)* and *under real (non-zero time) repair*.

6.1.2. Case studies results. Figure 4(a) plots, for the CRM system, the predicted total number of faults detected ($m_d(t)$) and corrected ($m_c(t)$) over time, and the difference between them.

First, the **sensitivity** of the difference is assessed with respect to variations of the detection and correction models. Figure 4(a) includes the correction and detection models obtained by *incrementing* and *decrementing* by 1%[††] the distribution parameters obtained in Section 5. For instance, for CRM at day 200, $m_d(t)$ varies between 599 and 611 under a $\pm 1\%$ variation of the exponential parameters $\hat{\alpha}$ and $\hat{\beta}$. Let us denote by $m_{d\pm}(t)$ and $m_{c\pm}(t)$ the detection and correction models obtained by varying the parameters: the difference varies between the lower and upper bounds computed as follows:

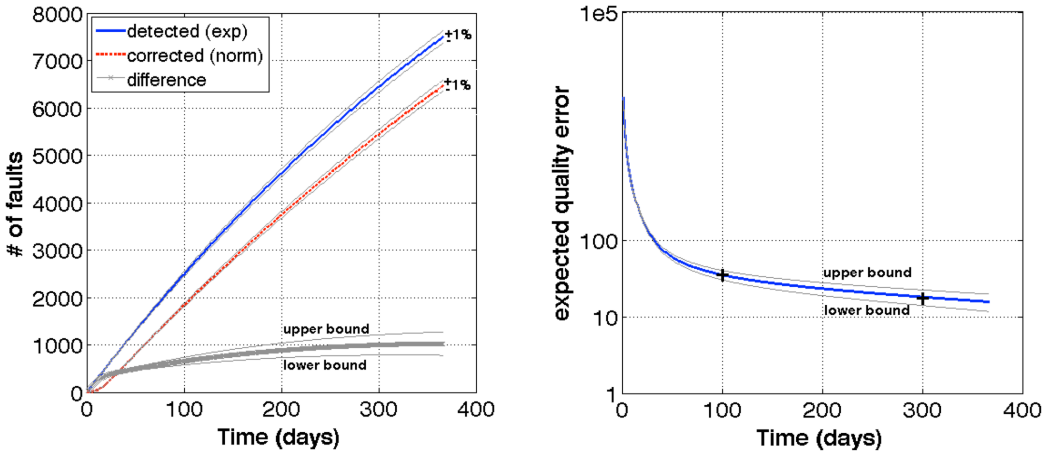
$$lower\ bound = m_{d-}(t) - m_{c+}(t), \qquad upper\ bound\ = m_{d+}(t) - m_{c-}(t). \tag{11}$$

Hence, for the CRM system – day 200 in Figure 4(a) – the difference varies between 18 and 42 when $m_d(t)$ and $m_c(t)$ are subject to a $\pm 1\%$ variation of the parameters.

As for the quality estimate, the largest difference between detection and correction models is observed in Figure 4(a) around day 100; then the models start flattening. Assume the product is planned for release when 75% of total expected faults (451 out of 602) are removed. Under the immediate repair assumption ($m_c(t) = m_d(t)$), 451 faults should have been corrected at day 48 according to the prediction model ($m_d(48) = 451$). However, considering the predicted trend of correction times, only 389 faults would actually be corrected at day 48, due to the latency of debugging ($m_c(48) = 389$). The impact on the prediction of the expected quality at day 48 ($\epsilon_Q(48)$), is about 16% according to Equation 10; in other words, using a SRGM assuming immediate repair

[††]The variation of the parameters is limited within $\pm 1\%$ as for larger ranges the detection model may assume values smaller than the correction model.

DOI: 10.1002/stvr



(a) Predicted number of detected and corrected faults. (b) Expected quality error (y-axis in logarithmic scale).

Figure 5. Impact on reliability estimate for the ERP system.

causes the overestimation of the actual expected product quality by 16%. In absolute terms, the biggest difference in the estimate is at day 14 (where $m_d(t)$ - $m_c(t)$ is 166 faults). The percentage error $\epsilon_Q(t)$ is greater at the earlier stages, where the difference between $m_d(t)$ - $m_c(t)$ is higher. Figure 4(b) shows how the expected quality error $\epsilon_Q(t)$ varies with time in the CRM case (along with lower/upper bounds). The points marked by '+' correspond to the above-discussed days 48 and 100. For instance, postponing the product release at day 100 would make it possible to detect 571 faults. Again, due to the debugging latency, only 542 out of these 571 faults are actually corrected at that day, which causes an overestimate of 5.3% under the SRGM immediate repair assumption. Figures 5(a) and 5(b) show similar results for the ERP system; *difference* and *expected quality error* are shown with the lower and upper bounds obtained by varying detection and correction model parameters by ±1%. The error on the expected quality drops from 36% to 18% from day 100 to 300 - marked by a '+' in Figure 5(b). Similarly to the CRM, the impact of the debugging time on the quality prediction decreases as the release time increases.

By knowing the trend of detection and correction times built on collected data, engineers can decide either to postpone the release time to actually get the desired quality, or to work on the debugging process to reduce the impact on the estimate at the established release time.

6.2. *Impact on release time prediction*

6.2.1. Metric. The correction process impacts the prediction of the release time inferred through SRGMs. Suppose the company wishes to determine the product release time ensuring a quality of 99% (i.e., the 99% of detected faults have been corrected). Under the immediate repair assumption the release time would be such that $t = \{t : \frac{m_d(t)}{m_d(\infty)} = 0.99\}$. In practice, due to the latency of debugging activities, 99% of faults are corrected only later at $t^* = \{t^* : \frac{m_c(t^*)}{m_d(\infty)} = 0.99\}$, with $t^* > t$: $\rho_T = (t^* - t)$ is the time to wait until 99% of detected faults are really corrected.

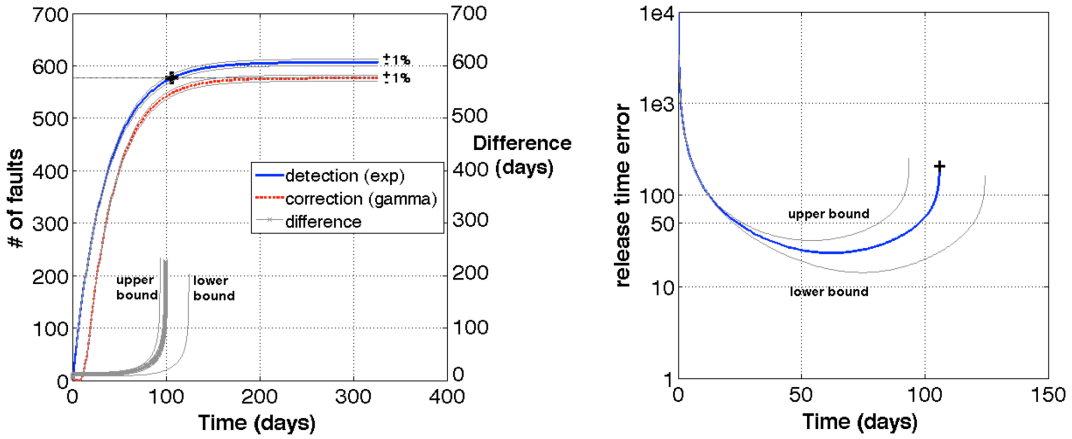
More generally, the time span needed to ensure that the number of faults detected at time t are also corrected is: $\{t^* : m_c(t^*) = m_d(t)\} - t$. The percentage error on the release time is:

$$\epsilon_T(t) = \frac{\{t^* : m_c(t^*) = m_d(t)\} - t}{t} \cdot 100. \tag{12}$$

$\epsilon_T(t)$ is the percentage difference between (i) the ***actual* time** t^* to achieve a given quality (number of detected *and* corrected faults), and (ii) the ***ideal* time** t for achieving that quality.

The prediction of the release time is of paramount importance. It is reasonable to state that software with uncorrected issues would not be released to users: our models allow predicting the *minimum time* an industry provider should wait, at that fault correction pace, in order to ensure that all the issues detected at a given time t are also corrected. This of course can trigger improvement actions, e.g., to expedite the process if release schedule requirements would not be met.

DOI: 10.1002/stvr



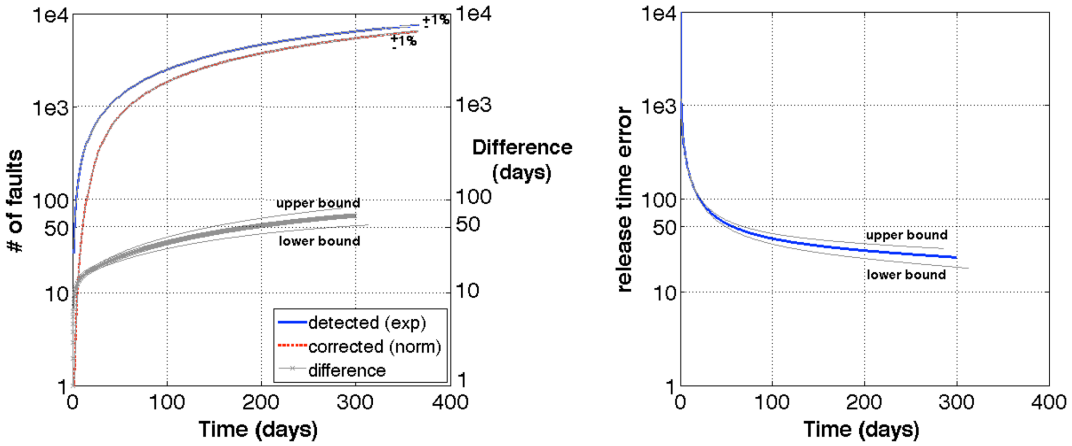
(a) Difference between actual and ideal release times. (b) Release time error (y-axis in logarithmic scale).

Figure 6. Impact on release time prediction for the CRM system.

6.2.2. Case studies results. Figures 6(a) and 7(a) plot the predicted total number of detected and corrected faults for the two case studies (left y-axis). In the CRM the maximum number of corrected faults, i.e., 577 (approximation of 576.5), is obtained at day 327; however, 577 faults were detected at day 106, as indicated by the data point marked by a '+' in Figure 6(a). Starting from day 106, it takes further $327 - 106 = 221$ days to ensure that the same number of detected faults are corrected. Under the immediate debugging assumption, the tester would release at day 106, rather than 327, to correct those 577 faults. This corresponds to an error equal to: $\epsilon_T(106) = \frac{327-106}{106} \cdot 100 = 208.5\%$ according to Eq. 12. Similarly, in the ERP system, it takes 67 days until the number of faults detected at day 299 (6,467) are also corrected, which is an error of: $\epsilon_T(366) = \frac{366-299}{299} \cdot 100 = 22.4\%$.

The difference between actual and ideal release times is plotted in Figures 6(a) and 7(a) too, *difference* series. For a day d, the plot indicates the number of days (y-axis on the right) needed to ensure that the same number of faults detected until d have also been corrected. Release time series are shown for both the case studies with lower and upper bounds, which visualize the range of values the release time assumes when correction and detection are subjected to $\pm 1\%$ variation.

Figures 6(b) and 7(b) show how the percentage error $\epsilon_T(t)$ varies over time and its lower/upper bounds. For instance, let us assume again to release the CRM system at day 100 (e.g., because of a constraint on the maximum release time). Under the immediate debugging assumption, releasing at day 100 yields an "ideal" quality of about 95% (at day 100, 571 faults are detected out of the



(a) Difference between actual and ideal release times. (b) Release time error (y-axis in logarithmic scale).

Figure 7. Impact on release time prediction for the ERP system.

DOI: 10.1002/stvr

total expected 602 faults: $(571/602) \cdot 100 = 94.8\%$). Thus, neglecting the latency of debugging causes the underestimation of the release time by $\frac{157-100}{100} \cdot 100 = 57.0\%$, because the actual 95% of quality is obtained at day 157 instead of day 100. Similarly, releasing the ERP at day 100 causes underestimating the release time by $\frac{134-100}{100} \cdot 100 = 34.0\%$. In fact, the assumption impacts not only the quality estimation/prediction, but clearly also the optimal release time determination.

6.3. *Impact on cost estimate*

6.3.1. Cost computation. SRGMs are used to minimize the overall *fault removal* and *testing* cost at time t. Similarly to [36, 46], DWA-SRGM adopts the cost function of Equation 13 to analyze the impact caused by the immediate debugging assumption on cost estimation:

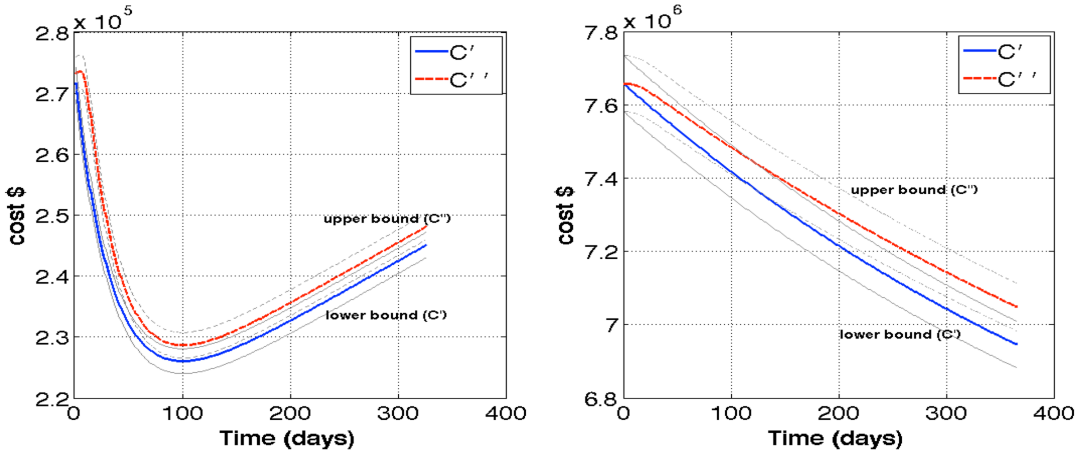
$$C' = c_1 \cdot m_d(t) + c_2 \cdot [m_d(\infty) - m_d(t)] + c_3 \cdot t \tag{13}$$

where: c_1 is the expected cost of removing a fault during testing; c_2 is the expected cost of removing a fault during the operation phase; c_3 is the expected cost per unit time for testing. The following values are assumed for cost factors: $c1 = 350\$$, $c2 = 450\$$, and $c3 = 100\$$; however, any other set of representative values would fit the scope of the analysis without impacting the findings discussed in the following. Considering the fault correction distribution, Equation 13 becomes:

$$C'' = c_1 \cdot m_c(t) + c_2 \cdot [m_d(\infty) - m_c(t)] + c_3 \cdot t \tag{14}$$

where $m_d(\infty) - m_c(t)$ is the number of uncorrected faults, which includes undetected faults, i.e., $m_d(\infty) - m_d(t)$, and detected but uncorrected faults, i.e., $m_d(t) - m_c(t)$.

6.3.2. Case study results. Figure 8(a) and 8(b) plot the cost functions C', C'' and related lower/upper bounds *vs* release time for the case studies. In the CRM the difference between C' and C'' increases as the release time increases, until it approaches day 100. Solution t_1^*, i.e., the optimal release time to minimize the expected cost, is day 103 according to C'; however, by considering Equation 14 (i.e., the cost function including the fault correction) the optimal release time would be $t_2^* = day\ 95$. The distance between the optimal release times is more than 1 week, as it can be also inferred by Figure 8. The percentage difference between t_1^* and t_2^*, i.e., $\epsilon_T = \frac{t_1^* - t_2^*}{t_2^*} \cdot 100 = 8.4\%$ when considering the optimal cost. Neglecting the fault correction process causes underestimation of the actual costs (C' is smaller than C'' for both CRM and ERP), and, as noted for CRM, missing the actual optimal release time.



(a) CRM. (b) ERP.

Figure 8. Predicted cost *vs* release time.

DOI: 10.1002/stvr

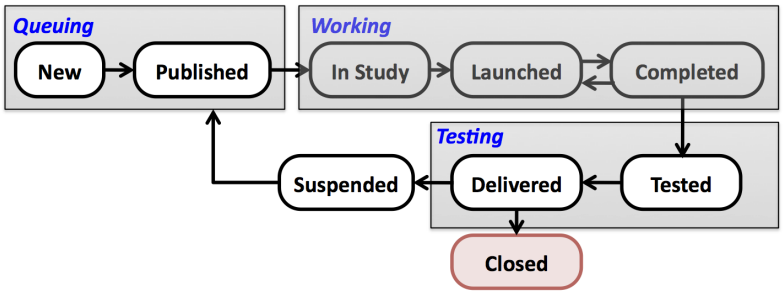


Figure 9. Workflow of the debugging process.

7. DEBUGGING-WORKFLOW-AWARE SRGM MODELING

This section presents the analysis of the impact of the debugging workflow on the quality estimation and release time prediction for the CRM case study. The **debugging workflow** is depicted in Figure 9. It has been devised from the states traversed by a bug record during its lifetime, as the example listed in Table VII for the issue "CRM-2". The states are grouped into three main phases:

- *Queuing*. Once a fault is detected, an issue is created (*new*) and enqueued (*published*);
- *Working*. The issue is dequeued and its state changes into *in_study*; it is then assigned to a debugger (*launched*); once *completed*, if undergoes testing;
- *Testing*. The amendment is *tested*, *delivered*, and finally *closed*.

As the test of an amendment may fail, an issue may become *suspended* after delivered, and is then re-opened for another cycle of processing (from the *published* state). The dataset contains also not closed issues, either because still under processing (for the CRM system, this happens for issues opened in January 2014) or because finally classified with a *won't_fix* resolution.

Table VII. Example of history for a CRM bug (date format: DD/MM/YY).

ID	Modified by	Date	Entry type	Old value	New Value
CRM-2	*operator_name_1*	07/09/12 10:16	Assignee	-	-
CRM-2	*operator_name_2*	07/09/12 12:52	Subcomponent(s)	-	Incomes Management
CRM-2	*operator_name_2*	07/09/12 12:52	Component(s)	-	Frontend
...					
CRM-2	*operator_name_2*	11/09/12 17:41	Status	New	Published
CRM-2	*operator_name_3*	17/09/12 10:12	Status	Published	In study
CRM-2	*operator_name_3*	17/09/12 10:39	Status	In study	Launched
CRM-2	*operator_name_3*	17/09/12 14:09	Status	Launched	Completed
CRM-2	*operator_name_3*	17/09/12 14:09	Status	Completed	Tested
CRM-2	*operator_name_4*	21/09/12 12:02	Status	Tested	Delivered
CRM-2	*operator_name_4*	21/09/12 12:02	Status	Delivered	Closed

7.1. Workflow statistics

Figure 10 shows the breakdown of issues by CRM functionality. The majority of issues affected the functionalities *Sales management* and *Customer folder*, consistently with their complexity and central role in the system. The *other* category contains 20 issues from 6 functionalities. The following statistics are computed for the tracked issues:

- *MTTR* (in days): mean time to repair (TTR, time interval from the *new* to the *closed* state);
- MED (in days): median of the TTR;
- $StdDev$ (in days): standard deviation of the TTR;
- *#Res*: average number of human resources allocated to the issue;
- *Kurtosis*: it indicates the peakedness of the distribution;
- *Skewness*: it measures the asymmetry of the distribution; a positive (negative) value indicates a right-tailed (left-tailed) distribution.

DOI: 10.1002/stvr

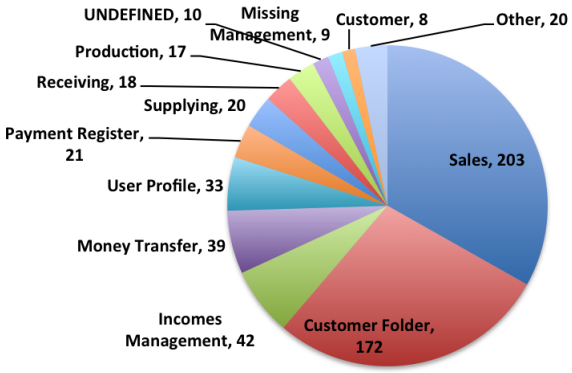


Figure 10. Breakdown of detected bugs per software functionality for the CRM system.

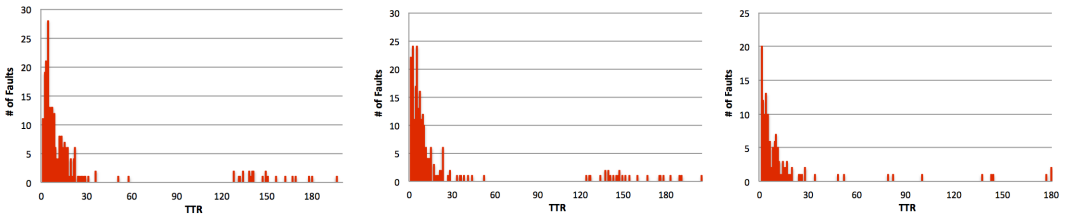
Table VIII shows the values, with issues grouped per severity. As expected, debugging is not immediate. The mean TTR on the whole dataset is as high as 12.8 days. The workflow exhibits desirable features: issues are closed in a reasonable time frame (from 10 to 16 days on average, median: 5-6 days), though highly variable (high StdDev). Moreover, *blocking* issues are solved faster than *major* ones, which in turn are solved faster than *minor* ones.

Table VIII. Statistics of the TTR distributions for CRM system (mean values and std deviations in days).

Bug Severity	*MTTR*	*MED*	*StdDev*	*#Res*	*Kurtosis*	*Skewness*
Minor	15.88	6.16	29.29	3.69	42.81	6.29
Major	12.69	5.76	25.28	3.75	52.26	6.85
Blocking	9.85	4.70	21.42	3.72	79.89	8.24

Further hints are provided by the kurtosis and skewness indicators. A high value of kurtosis denotes that most of the variance is due to few high peaks, as opposed to the undesirable situation of frequent small deviations from the mean time to repair. In a plot of the distribution, it is desirable that these high peaks are concentrated in the left side, as this would indicate that many faults have a short TTR. Moreover, it is desirable for the distributions to be right-tailed, as this would indicate that most defects have a low time to repair; this corresponds to a positive value of skew.

The TTR distributions for the three severity classes for the CRM system are shown in Figure 11: they are all right-tailed with pronounced peaks in the left side. On average, 3.7 debuggers are allocated per issue, regardless of the severity. Overall, this debugging process presents good characteristics; nonetheless, its impact on reliability analyses accuracy can mislead decision makers.



(a) TTR for *minor* faults (b) TTR for *major* faults (c) TTR for *blocking* faults

Figure 11. TTR distributions for *minor*, *major* and *blocking* faults for the CRM system.

7.2. Modeling

A fault correction model $m_c(t)$ is built for each **workflow phase**, adopting the same procedure used for the fault correction process characterization (namely, fitting the *exponential*, *normal* and *gamma* distributions and selecting the one with the lowest *%RPE* and, secondary, highest R^2). The *queuing* distribution considers the queuing times (rather than the whole correction times), and

DOI: 10.1002/stvr

represents the impact of the *queueing* phase on the mean value function. Using the *queuing+working* times, we characterize the correction process in the ideal case of all fixes having zero testing time. Finally, comparing the *total* correction times to *queuing+working* or to *queueing* times accounts for the impact of the bug fix *testing* or *working+testing* phase, respectively. This allows studying the weight of each debugging phase, and the error committed if the estimations of such phases were neglected. Table IX lists data by cumulative debugging time (*queueing*; *queueing+working*; *queueing+working+testing*, which corresponds to the *total* correction time reported in Table V).

Table IX. Model parameters per phase for the CRM system (Correction).

Model	Parameters				R^2	*%RPE*
Queuing						
Exp	$\hat{\alpha} = 590.91$	$\hat{\beta} = 0.055$	$\hat{\gamma} = 0.055$	-	0.993	-2.89%
Norm	$\hat{\alpha} = 646.37$	$\hat{\beta} = 0.025$	$\hat{\mu} = 0.122$	$\hat{\sigma} = 0.083$	0.968	-3.59%
Gamma	$\hat{\alpha} = 591.49$	$\hat{\beta} = 0.040$	$\hat{\mu} = 1.768$	$\hat{\kappa} = 7.035$	0.994	-1.17%
Queuing + Working						
Exp	$\hat{\alpha} = 582.64$	$\hat{\beta} = 0.036$	$\hat{\gamma} = 0.081$	-	0.982	-6.28%
Norm	$\hat{\alpha} = 596.06$	$\hat{\beta} = 0.023$	$\hat{\mu} = 0.140$	$\hat{\sigma} = 0.055$	0.961	-6.84%
Gamma	$\hat{\alpha} = 582.87$	$\hat{\beta} = 0.036$	$\hat{\mu} = 4.419$	$\hat{\kappa} = 2.510$	0.984	-1.37%
Queuing + Working + Testing (i.e., Total)						
Exp	$\hat{\alpha} = 577.03$	$\hat{\beta} = 0.033$	$\hat{\gamma} = 0.101$	-	0.971	-0.42%
Norm	$\hat{\alpha} = 591.61$	$\hat{\beta} = 0.023$	$\hat{\mu} = 0.133$	$\hat{\sigma} = 0.056$	0.954	1.76%
Gamma	$\hat{\alpha} = 576.54$	$\hat{\beta} = 0.032$	$\hat{\mu} = 9.704$	$\hat{\kappa} = 1.032$	0.974	-0.19%

The same approach is used for fine-grained analysis with respect to defect **severity** and impacted product **functionality**. Table X shows the parameters of the correction models by severity. Table XI shows the top-5 functionalities by number of faults (accounting for 79.90% of total defects). In all but three cases Gamma is the selected best fitting model for the fault correction process; in three cases the Exponential model is selected, with a *%RPE* always under 3% and an R^2 always greater than 0.95. For instance, the model *Blocking* of Table X represents the ideal case in which debuggers performed for all other faults as they did for the *Blocking* faults; in the same way, the model *Sales* represents the case of debuggers performing always as for the *Sales* functionality. These distributions are now used to analyze the impact of bug features on the estimation/prediction of delivered product quality, and on the optimal release determination (time and cost).

Table X. Model parameters per severity for the CRM system (Correction).

Model	Parameters				R^2	*%RPE*
Minor						
Exp	$\hat{\alpha} = 218.83$	$\hat{\beta} = 0.058$	$\hat{\gamma} = 0.058$	-	0.987	-6.56%
Norm	$\hat{\alpha} = 240.32$	$\hat{\beta} = 0.027$	$\hat{\mu} = 0.118$	$\hat{\sigma} = 0.082$	0.950	-8.74%
Gamma	$\hat{\alpha} = 215.54$	$\hat{\beta} = 0.047$	$\hat{\mu} = 4.649$	$\hat{\kappa} = 2.679$	0.993	-4.45%
Major						
Exp	$\hat{\alpha} = 220.54$	$\hat{\beta} = 0.034$	$\hat{\gamma} = 0.095$	-	0.965	-14.08%
Norm	$\hat{\alpha} = 236.00$	$\hat{\beta} = 0.050$	$\hat{\mu} = 0.100$	$\hat{\sigma} = 0.100$	0.754	-11.77%
Gamma	$\hat{\alpha} = 220.46$	$\hat{\beta} = 0.033$	$\hat{\mu} = 9.206$	$\hat{\kappa} = 1.104$	0.968	-4.82%
Blocking						
Exp	$\hat{\alpha} = 124.44$	$\hat{\beta} = 0.034$	$\hat{\gamma} = 0.143$	-	0.984	-8.14%
Norm	$\hat{\alpha} = 202.50$	$\hat{\beta} = 0.026$	$\hat{\mu} = 0.046$	$\hat{\sigma} = 0.149$	0.972	-8.15%
Gamma	$\hat{\alpha} = 124.46$	$\hat{\beta} = 0.034$	$\hat{\mu} = 4.802$	$\hat{\kappa} = 1.424$	0.985	-1.32%

7.3. Reliability estimate

Debugging phase. The differences are now computed between the predicted number of detected faults and the predicted number of faults that have reached the end of the *i)* queuing phase, *ii)*

DOI: 10.1002/stvr

Table XI. Model parameters per functionality for the CRM system (Correction).

Model	Parameters				R^2	%*RPE*
Sales						
Exp	$\hat{\alpha} = 196.82$	$\hat{\beta} = 0.052$	$\hat{\gamma} = 0.052$	-	0.989	-2.87%
Norm	$\hat{\alpha} = 198.83$	$\hat{\beta} = 0.025$	$\hat{\mu} = 0.136$	$\hat{\sigma} = 0.055$	0.943	-3.94%
Gamma	$\hat{\alpha} = 197.67$	$\hat{\beta} = 0.031$	$\hat{\mu} = 0.210$	$\hat{\kappa} = 32.698$	0.961	-3.26%
Customer Folder						
Exp	$\hat{\alpha} = 159.89$	$\hat{\beta} = 0.039$	$\hat{\gamma} = 0.124$	-	0.962	-10.75%
Norm	$\hat{\alpha} = 192.88$	$\hat{\beta} = 0.027$	$\hat{\mu} = 0.101$	$\hat{\sigma} = 0.101$	0.946	-11.24%
Gamma	$\hat{\alpha} = 159.71$	$\hat{\beta} = 0.039$	$\hat{\mu} = 9.855$	$\hat{\kappa} = 0.848$	0.966	-2.32%
Incomes Management						
Exp	$\hat{\alpha} = 38.36$	$\hat{\beta} = 0.054$	$\hat{\gamma} = 0.112$	-	0.987	-1.43%
Norm	$\hat{\alpha} = 58.72$	$\hat{\beta} = 0.034$	$\hat{\mu} = 0.058$	$\hat{\sigma} = 0.137$	0.971	-9.81%
Gamma	$\hat{\alpha} = 39.65$	$\hat{\beta} = 0.040$	$\hat{\mu} = 0.210$	$\hat{\kappa} = 24.890$	0.975	-4.46%
Money Transfer						
Exp	$\hat{\alpha} = 32.43$	$\hat{\beta} = 0.050$	$\hat{\gamma} = 0.170$	-	0.983	-17.21%
Norm	$\hat{\alpha} = 35.72$	$\hat{\beta} = 0.037$	$\hat{\mu} = 0.118$	$\hat{\sigma} = 0.084$	0.972	-2.87%
Gamma	$\hat{\alpha} = 32.41$	$\hat{\beta} = 0.054$	$\hat{\mu} = 0.689$	$\hat{\kappa} = 10.032$	0.983	0.32%
User Profile						
Exp	$\hat{\alpha} = 29.04$	$\hat{\beta} = 0.065$	$\hat{\gamma} = 0.791$	-	0.959	-2.16%
Norm	$\hat{\alpha} = 58.31$	$\hat{\beta} = 0.060$	$\hat{\mu} = 0.003$	$\hat{\sigma} = 0.178$	0.957	14.78%
Gamma	$\hat{\alpha} = 32.59$	$\hat{\beta} = 0.039$	$\hat{\mu} = 9.869$	$\hat{\kappa} = 0.000$	0.724	18.56%

working phase (*queuing+working*), and *iii)* entire debugging (*queuing+working+testing*, i.e., *all phases*). Results are shown in Figure 12(a)[‡‡].

The overall difference (*all phases* series) can be attributed to individual phases as follows. Let us consider the day where 60% of quality under the immediate debugging assumption is achieved, which is day 33 when 361 out of 602 faults are detected. At this day, the difference in the *all phases* series is 96 faults (data points marked by $\times$ in Figure 12(a)). Out of these 96 bugs, *(i)* 48 are in the *queuing* phase; *(ii)* 22 (i.e., 70-48, with 70 denoting the value of *queuing+working* on day 33) are in the working phase; *(iii)* 26 (i.e., 96-70) are in testing phase. As such, on day 33 the 50% of the difference is due to queueing ($\frac{48}{96} \cdot 100$); working and testing account for the 23% and 27% of the difference, respectively. This means that if engineers want an accurate estimate at day 33 (or, in a specular way, they want to achieve the *actual* 60% as earlier as possible beyond day 33), they need to improve mostly the queueing time (e.g., improving the assignment of bugs to debuggers).

On average, most of the pending faults are faults waiting in the queue. Based on such models, this kind of analysis can be conducted referring to past times (to estimate the impact of a phase up to time t) and to future times (to predict the impact and possibly take corrective actions).

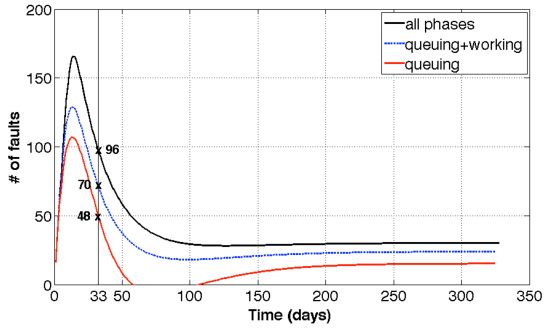
Severity. Figure 12(b) shows the difference between the predicted number of detected and corrected faults for *minor*, *major*, *blocking* severities. The parameters of the correction distributions are reported in Table X.

The *minor* and *major* distributions are really close between days 1 and 50: their impact on the expected quality is almost the same. The difference between detected and corrected faults is significantly smaller for the *blocking* severity. The maximum difference is 63 faults (at day 14) and 28 faults (at day 11) in the case of *major* and *blocking* faults, respectively. This means that if all the faults were corrected as the *blocking* ones, the immediate repair assumption would have caused a smaller distortion on the estimate of the expected quality. The finding is consistent with the TTR-related statistics shown in Table VIII, where it can be noted that *blocking* faults are characterized by the lowest *MTTR* and *MED* values when compared to *major* and *minor* faults.

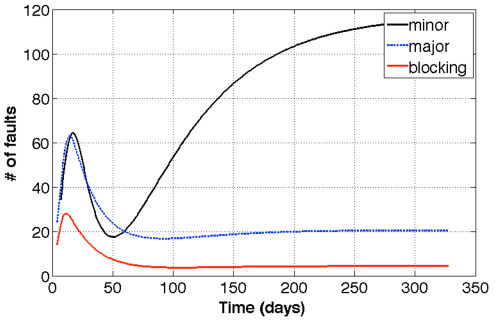
Note that the difference between detected and corrected *minor* faults does not converge to an asymptotic value, as it can be inferred from Figure 12(b) (*minor* series). This is a rather anomalous scenario, which is explained by a closer analysis of *minor* faults.

[‡‡]Note that the plots may go below zero, since these are differences *predicted* by models: a fault correction model by phase may overcome the detection model at some points where the raw data series are very close to each other.

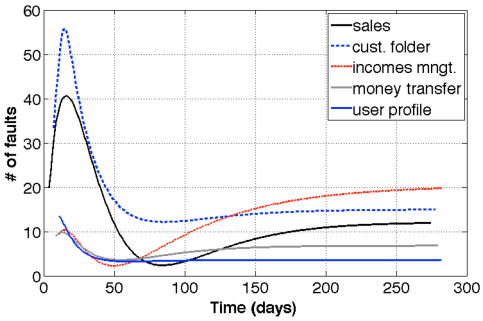
DOI: 10.1002/stvr



(a) Debugging Phase



(b) Severity



(c) Functionality

Figure 12. Difference between the predicted number of detected and corrected faults by phase, severity and functionality.

Figure 13 shows the actual/predicted number of detected/corrected *minor* faults. We observed that no fault is corrected between day 78 and day 197 (i.e., the cumulative number of corrected faults remains 212 over this timeframe), which misleads the predicted difference. The debugging team of the industrial partner confirmed that the faults were not promptly corrected because they affected a software module excluded from a prior product release and re-integrated later on.

Functionality. The difference between predicted number of detected/corrected faults by *functionality* (Figure 12(c)) indicates that *Customer folder* and *Sales* are the ones most impacted by the immediate debugging assumption. In fact, the debugging delay for them is significantly larger than for the other functions, with the maximum difference observed around day 15.

The industrial partner confirmed that they both represent complex and fault-prone modules of the CRM. The analysis suggests that if all faults were corrected as by the teams managing the *Incomes Management*, *Money Transfer* and *User Profile* functionalities, neglecting the non-immediate debugging has a lower impact on the error about quality prediction. If this kind of

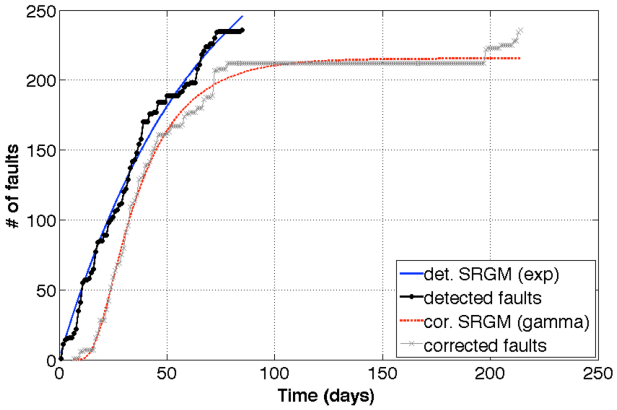


Figure 13. Detection/correction of *minor* faults over time.

DOI: 10.1002/stvr

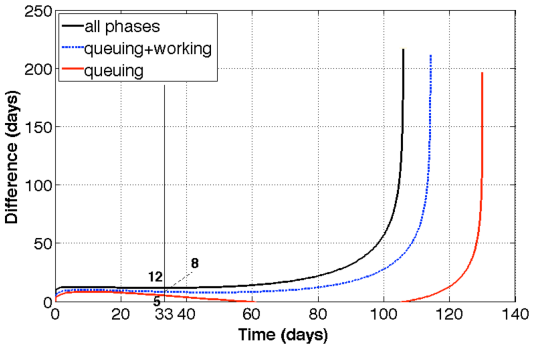
information is used during the process, engineers can improve debugging of those functionalities expected to impact more on the error (e.g., moving people from one functionality to another).

7.4. *Release time prediction*

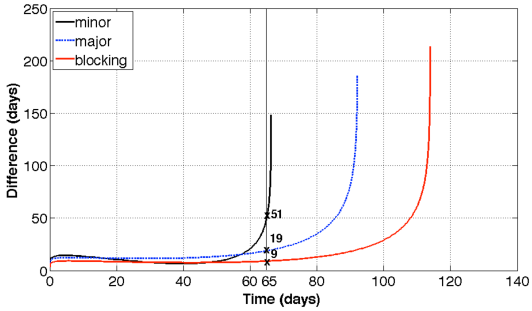
The models described in Section 7.2 allow to discriminate the contribution of *debugging phases*, bug *severity* and system *functionality* on the release time prediction. Figures 14 shows the differences between actual and ideal release time for such factors.

Debugging phase. Let us consider again day 33, when 361 out of 602 faults are detected. The total difference between actual and ideal release time on day 33 is 12 (Figure 14(a) - *all phases* series), which means it takes 12 days until the number of faults detected on day 33 is also corrected. The *queueing* and *queueing+working* series make it possible to brake down the total difference by phase. Figure 14(a) - *queuing* series - indicates that on day 33, the above-mentioned 361 faults are out of the queuing phase after 5 days since their detection. By including the working phase, the difference becomes 8 (*queuing+working* series), which means that it takes further 3 days (8-5) until the 361 faults are out of working. Finally, the inclusion of the testing phase makes the difference to become 12 (*all phases* series), which means that testing accounts for further 4 days (12-8) of the total release time difference. As a result, on day 33 queuing contributes to the 42% of the release time difference ($\frac{5}{12} \cdot 100$), while working and testing to the 25% and 33%, respectively. On average, *queuing* is the phase that mostly undermines the immediate debugging assumption.

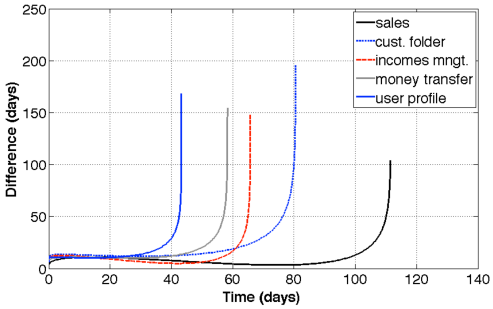
Severity. The analysis suggests that severity values contribute to rather different extent to the total difference between actual and ideal release time. For example, while the total difference is about 15 days on day 65, it ranges from 9 to 51 when the faults are broken down by severity ($\times$-marked data points in Figure 14(b)). *Blocking* faults are characterized by the lowest difference with respect to time: this finding suggests that duration of the debugging workflow of *blocking* faults is smaller than *major* and *minor*, which is also supported by the *MTTR*s shown in Table VIII.



(a) Debugging phase



(b) Severity



(c) Functionality

Figure 14. Impact of factors on actual/ideal release time for the CRM system.

DOI: 10.1002/stvr

Functionality. The functionality impacts the release time significantly. For example, we found out that the difference between actual and ideal release time of *User profile* becomes large after 40 days; in case of *Sales* it takes about 100 days before the difference becomes significant. Noteworthy, the faster fault detection goes to saturation, the earlier the difference between actual/ideal release time increases. Figure 15 shows the fault detection processes of *Sales* and *User profile*. *Sales* saturates around day 80, while user profile around day 25: this is consistent with the difference between actual/ideal release time, which starts increasing after day 80 and 25 in sales and user profile, respectively (Figure 14(c)).

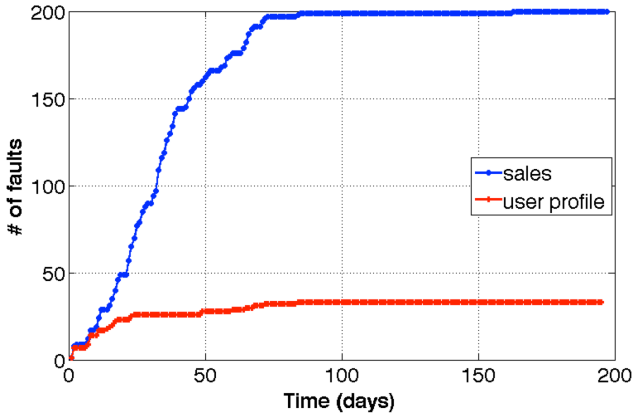


Figure 15. The actual fault detection process of the *Sales* and *User profile* CRM functionalities.

8. DISCUSSION

8.1. *Key features of the proposed method*

We have presented a method for software reliability growth modeling, which properly accounts for the workflow of debugging. The method leverages the defect data usually managed by software companies by means of issue trackers. We have shown its application to two real-world software projects. The study indicates that assuming immediate debugging may cause overestimation of the product quality achievable at a target time, and relevant errors in release time planning. This increases the risk of releasing software with more residual defects than expected, with non-negligible impact on maintenance costs, customer dissatisfaction and company reputation.

The experiments show that SRGM-based estimates/predictions are significantly influenced by high variability of the duration of debugging activities. Moreover, SRGMs considering actual debugging times provide more accurate predictions of the costs.

The availability of detailed data from issue trackers allows to discriminate precisely the factors with greater influence on quality and release time, and to undertake proper process improvement actions. This way, software process managers are supported in taking informed decision concerning:

- The allocation of professionals to debugging where actually needed (e.g., to bottleneck functionalities, or to classes of bugs depending on their severity);
- The tuning of debugging activities aimed at: *i)* lowering the average correction time; *ii)* reducing differences among debugging teams; *iii)* reducing idle times (queue of the bug assignments); *iv)* the treatment of bugs of different types (e.g., of different severity).

For instance, in both the CRM and ERP case study, the method made it possible to quantify the impact of the debugging time on quality and release time estimates; moreover, in the CRM the method allowed to identify a bottleneck of the workflow in the queuing phase of bugs to debuggers: being excessively long and variable, this phase is the major cause of prediction error.

DOI: 10.1002/stvr

8.2. Threats to validity

The analysis conducted in this study was possible thanks to the availability of detailed data collected in the issue tracking system. The inferences made on a dataset are subject to both construct and internal validity threats. Bug data are not always recorded with the needed care, or, even worse, data collection might be overlooked, leading to mistakes (e.g., test engineers often update records only when the scheduled release time is approaching, not when the issues are actually closed). This is a potential threat that practitioners replicating this type of analysis should be aware of.

A further threat concerns the selection of the SRGM. In general, there is no SRGM that fits all situations; we decided to not rely on a predefined one, but to adopt four models and choose the best fitting one. Of course, other SRGMs could work better for a given dataset; practitioners are always called to select the SRGMs best suited for their testing process data.

Further considerations derive from severity analysis. These are valid under the assumption that developers within the company use the same criteria to assign severity to issues, and the bias due to different people doing this task is minimized. More generally, results could be not representative of the context if people involved in the analyzed process were aware of the study; however, we assured that neither developers nor testers of the analyzed system were aware of the study – hence results are free from potential psychological bias.

As for external validity, results observed on a single case are clearly not statistically generalizable. The reported findings, supported by data, may serve as basis to develop more precise approaches considering all facets of debugging in SRGM-based reliability analysis. Since similar analyses on proprietary industrial systems are scarce in the literature, we believe that the results we reported are meaningful for practitioners despite the limitations in external validity.

9. CONCLUSION

This paper has presented DWA-SRGM, a method combining analytical modeling and empirical assessment for data-driven software reliability growth analysis during advanced testing phases. DWA-SRGM leverages data typically available to software companies in their issue tracking systems to improve SRGM predictions/estimates, and to support decisions on improvements of the debugging process.

The mathematical SRGM modeling framework, fed with data easily extracted from issue trackers, produces much more accurate predictions (of optimal release time) and estimates (of achieved product reliability at release) than those typically made under the non-realistic assumption of immediate repair. It also supports fine-grained analysis of the impact on SRGM-based estimates of bug features and debugging workflow activities; this gives managers the opportunity to spot bottlenecks and improve the process.

The results with two real-world case studies show that debugging times actually cannot be considered negligible, as they significantly affect the estimation of reliability figures. They revealed to be very valuable for the software process managers, *i)* to improve the accuracy of SRGM-based predictions, with a direct impact on key decisions about effort allocation and release time scheduling; *ii)* to have a quantitative feedback on the debugging process, taking into account the impacted software functionalities, the type of bugs, and the phases of the debugging process that need more attention for cost-effective quality improvement. This pragmatic approach supports a more effective use of SRGM-based software reliability analysis in industrial practice.

ACKNOWLEDGEMENTS

This work has been partially supported by European Union under project ICEBERG (grant no. 324356) of the Marie Curie IAPP program, and by MIUR within the GAUSS project (CUP E52F16002700001) of the PRIN 2015 program.

DOI: 10.1002/stvr

REFERENCES

1. D. R. Jeske and X. Zhang, "Some successful approaches to software reliability modeling in industry," *Journal of Systems and Software*, vol. 74, no. 1, pp. 85–99, 2005.
2. R. Rana, M. Staron, N. Mellegård, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Evaluation of Standard Reliability Growth Models in the Context of Automotive Software Systems," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 7983, pp. 324–329.
3. C. Huang and M. Lyu, "Optimal release time for software systems considering cost, testing-effort, and test efficiency," *IEEE Trans. on Reliability*, vol. 54, no. 4, 2005.
4. C. Stringfellow and A. A. Andrews, "An empirical method for selecting software reliability growth models," *Empirical Software Engineering*, vol. 7, no. 4, 2002.
5. J.-H. Lo and C.-Y. Huang, "An integration of fault detection and correction processes in software reliability analysis," *Journal of Systems and Software*, vol. 79, no. 9, pp. 1312–1323, 2006.
6. P. K. Kapur, H. Pham, S. Anand, and K. Yadav, "A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation," *IEEE Trans. on Reliability*, vol. 60, no. 1, pp. 331–340, 2011.
7. C.-Y. Huang and W.-C. Huang, "Software reliability analysis and measurement using finite and infinite server queueing models," *IEEE Trans. on Reliability*, vol. 57, no. 1, pp. 192–203, 2008.
8. T. T. Nguyen, T. Nguyen, E. Duesterwald, T. Klinger, and P. Santhanam, "Inferring developer expertise through defect analysis," in *Proc. 34th Int. Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 1297–1300.
9. F. Zhang, F. Khomh, Y. Zou, and A. Hassan, "An Empirical Study on Factors Impacting Bug Fixing Time," in *Proc. 19th Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society, 2012, pp. 225–234.
10. A. Ihara, M. Ohira, and K. Matsumoto, "An analysis method for improving a bug modification process in open source software development," in *Proc. of the joint Int. and Annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol)*, 2009, pp. 135–144.
11. Bugzilla defect tracking system. [Online]. Available: www.bugzilla.org
12. JIRA Software. [Online]. Available: www.atlassian.com/software/jira
13. F. Febrero, C. Calero, and M. Angeles Moraga, "Software reliability modeling based on ISO/IEC SQuaRE," *Information and Software Technology*, vol. 70, pp. 18–29, 2016.
14. ——, "A Systematic Mapping Study of Software Reliability Modeling," *Information and Software Technology*, vol. 56, pp. 839–849, 2014.
15. M. Xie, "Software reliability models — A selected annotated bibliography," *Software Testing, Verification and Reliability*, vol. 3, no. 1, pp. 3–28, 1993.
16. A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. on Reliability*, vol. R-28, no. 3, pp. 206–211, 1979.
17. A. L. Goel, "Software reliability models: Assumptions, limitations and applicability." *IEEE Trans. on Software Engineering*, vol. SE-11, no. 12, pp. 1411–1423, 1985.
18. S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Trans. on Reliability*, vol. R-32, no. 5, pp. 475–484, 1983.
19. S. Gokhale and K. Trivedi, "Log-logistic software reliability growth model," in *Proc. 3rd Int. High-Assurance Systems Engineering Symposium (HASE)*, 1998, pp. 34–41.
20. K. Ohishi, H. Okamura, and T. Dohi, "Gompertz software reliability model: Estimation algorithm and empirical validation," *Journal of Systems and Software*, vol. 82, no. 3, pp. 535–543, 2009.
21. K. Kanoun, M. Kaâniche, J. Laprie, and S. Metge, "SoRel: A tool for reliability growth analysis and prediction from statistical failure data," in *Twenty-Third Int. Symposium on Fault-Tolerant Computing (FTCS), Digest of Papers*. IEEE, 1993, pp. 654–659.
22. K. Shibata, K. Rinsaka, and T. Dohi, "PISRAT: Proportional Intensity-Based Software Reliability Assessment Tool," in *Proc. 13th Pacific Rim Int. Symposium on Dependable Computing (PRDC)*. IEEE, 2007, pp. 43–52.
23. M. Lyu and A. Nikora, "CASRE: A computer-aided software reliability estimation tool," in *Fifth Int. Workshop on Computer-Aided Software Engineering*, 1992.
24. C. Huang, S. Kuo, and M. Lyu, "An assessment of testing-effort dependent software reliability growth models," *IEEE Trans. on Reliability*, vol. 56, no. 2, pp. 198–211, 2007.
25. R. Pietrantuono, S. Russo, and K. Trivedi, "Software reliability and testing time allocation: An architecture-based approach," *IEEE Trans. on Software Engineering*, vol. 36, no. 3, pp. 323–337, May 2010.
26. G. Carrozza, R. Pietrantuono, and S. Russo, "Dynamic test planning: a study in an industrial context," *International Journal on Software Tools for Technology Transfer*, vol. 16, no. 4, pp. 593–607, 2014.
27. B. Yang, Y. Hu, and C.-Y. Huang, "An Architecture-Based Multi-Objective Optimization Approach to Testing Resource Allocation," *IEEE Trans. on Reliability*, vol. 64, no. 1, pp. 497–515, 2015.
28. V. Almering, M. V. Genuchten, G. Cloudt, and P. Sonnemans, "Using Software Reliability Growth Models in Practice," *IEEE Software*, vol. 24, no. 6, pp. 82–88, 2007.
29. N. F. Schneidewind, "Analysis of error processes in computer software," *Sigplan Notices*, vol. 10, pp. 337–346, 1975.
30. M. Xie and M. Zhao, "The Schneidewind software reliability model revisited," in *Proc. 3rd Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 1992, pp. 184–192.
31. N. F. Schneidewind, "Modelling the fault correction process," in *Proc. 12th Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2001, pp. 185–190.
32. M. A. Stutzke and C. S. Smidts, "A stochastic model of fault introduction and removal during software development," *IEEE Trans. on Reliability*, vol. 50, no. 2, pp. 184–193, 2001.
33. S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, "Analysis of software fault removal policies using a non-homogenous continuous time markov chain," *Software Quality Journal*, vol. 12, no. 3, pp. 211–230, 2004.

DOI: 10.1002/stvr

34. J.-W. Ho, C.-C. Fang, and Y.-S. Huang, "The determination of optimal software release times at different confidence levels with consideration of learning effects," *Software Testing, Verification and Reliability*, vol. 18, no. 4, pp. 221–249, 2008.
35. J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability, Measurement, Prediction and Application*. McGraw Hill, 1987.
36. Y. Wu, Q. Hu, M. Xie, and S. Ng, "Modeling and analysis of software fault detection and correction process by considering time dependency," *IEEE Trans. on Reliability*, vol. 56, no. 4, pp. 629–642, 2007.
37. G. Carrozza, R. Pietrantuono, and S. Russo, "Defect analysis in mission-critical software systems: a detailed investigation," *Journal of Software: Evolution and Process*, vol. 27, no. 1, pp. 22–49, 2015.
38. M. Cinque, C. Gaiani, D. De Stradis, A. Pecchia, R. Pietrantuono, and S. Russo, "On the impact of debugging on software reliability growth analysis: A case study," in *Computational Science and Its Applications – ICCSA 2014*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8583, pp. 461–475.
39. R. Peng, Y. Li, W. Zhang, and Q. Hu, "Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction," *Reliability Engineering and System Safety*, vol. 126, pp. 37–43, 2014.
40. R. Mullen, "The Lognormal Distribution of Software Failure Rates: Application to Software Reliability Growth Modeling," in *Proc. 9th Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 1998, pp. 134–142.
41. H. Okamura, T. Dohi, and S. Osaki, "Software reliability growth model with normal distribution and its parameter estimation," in *Proc. Int. Conf. on Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE)*. IEEE, 2011, pp. 411–416.
42. H. Okamura, Y. Watanabe, and T. Dohi, "An iterative scheme for maximum likelihood estimation in software reliability modeling," in *Proc. 14th Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2003, pp. 246–256.
43. P. Kapur, D. Goswami, A. Bardhan, and O. Singh, "Flexible software reliability growth model with testing effort dependent learning process," *Applied Mathematical Modelling*, vol. 32, no. 7, pp. 1298–1307, 2008.
44. Y. Wu, Q. Hu, and S. Ng, "A Study of Software Fault Detection and Correction Process Models," in *Proc. Int. Conference on Management of Innovation and Technology*. IEEE, 2006, pp. 812–816.
45. T. Kima, K. Leeb, and J. Baik, "An effective approach to estimating the parameters of software reliability growth models using a real-valued genetic algorithm," *Journal of Systems and Software*, vol. 102, pp. 134–144, 2015.
46. M. Xie, *Software Reliability Modeling*. World Scientific Publishing, 1991.

DOI: 10.1002/stvr