Available online at: https://proceeding.researchsynergypress.com/index.php/rsfconferenceseries1
RSF Conference Series: Business, Management, and Social Sciences
e-ISSN 2807-5803/p-ISSN 2807-6699
Volume 2 Number 1 (2022): 218-230

A Proposal on How to Use Software Reliability Growth Model to Build Confidence in Dashboard Testing

Alexander Christian[1], Christopher Wibowo[1], Indriati Bisono[1], Hanijanto Soewandi[2]

[1]International Business Engineering Program, Universitas Kristen Petra, Surabaya, Indonesia

[2]MicroStrategy, Tysons Corner, VA, USA

Abstract

Software testing as an integral part of software development leads to the question of when the software (or application/dashboard) can be released and how confident that most defects/bugs/faults have been discovered. This paper discussed a relatively new but simple and practical proposal that can be used to build confidence for releasing software (or applications/dashboards). Instead of contrasting various software reliability growth models (SRGM) and choosing which one is the best, we use them to collaborate to help make decisions. We demonstrate our proposal with 18 real-life datasets that are publicly available in the literature. We use three widely used SRGMs, namely: Bass, Gompertz, and Logistic in our proposal to identify when we can stop testing. It turns out that when the testing has found most defects, most (if not all) of the SRGMs will converge to similar value for the maximum potential defects in the system.

Keywords: *SRGM, Bass, Gompertz, Logistic, Practical Stopping Rule.*

INTRODUCTION

Recently, software testing becomes an integral part of software development process and software engineering. While commercial company usually does not follow a rigid “defense and aerospace” software development process, all usually have their own software development life cycle, together with functional (as well as non-functional) requirements, specification, design and code review process, etc. Regardless the organization, the ultimate questions that executive wants to answer are usually: when a dashboard (or an application/software) can be released and how confident that most defects/bugs have been discovered during testing/development, i.e., some kind of metrics for the decision maker to make a good and informed decision. Please note that often times an executive can decide to release a software with known bugs (for various reasons, including: no more time to develop or no more defect in the system). Regardless, the above two questions are very hard to answer since there is no a priori knowledge about how many defects are in the system (software).

Even though there is no a priori knowledge about the number of defects in the software and the number of defects can vary from few hundreds to thousands, almost all researchers believed that the total defects in a software is finite. Given the rise of software reliability concept that have been developed in the past 50+ years, Cusick (2019) give an overview of Software Reliability Engineering (SRE) history.

According to Wood (1996) there are two types of software reliability models:

1. Defect density model – those that attempt to predict software reliability from the design parameters;
2. Software reliability growth model (SRGM) – those attempt to predict software reliability from test data.

Corresponding author
christ.alexander45@gmail.com

These models attempt to statistically correlate defects detection data over time with known functions such as: Gompertz, Logistics, Bass, etc. The focus in this paper is the second type.

To predict how many potential defects that exist in a software, we used 3 known functions, namely: Bass Diffusion – Innovation (Bass for short), Gompertz, and Logistics. We use empirical data to demonstrate how decision maker can gain confidence in making decision to declare the general availability of the software (or product/application) that is being tested. We propose very simple rules, yet practical, usable, and very intuitive – in particular, for high level executive for decision - making.

LITERATURE REVIEW

The use of Bass, Gompertz, and/or Logistics in SRGM is not new. Many researchers have used them previously, e.g., Kapur et.al. (2006), Chakravarty (2007), Shaik and Akthar (2011), Gandhi et.al. (2019), and very recently Yaghoobi (2021) as well as Haque & Ahmad (2021).
Most (if not all) research we found focused on how good a model can fit in predicting the growth of the software reliability. They compared various models to see which model provide the best fit. Surprisingly, many still use R-square (or Adjusted R-square) as a possible criterion in evaluating models even though Spiess and Neumeyer (2010) have demonstrated some problems with R-squared for non-linear least square via Monte Carlo simulation. We don't intend to compare various models. In fact, we recommend for practical reason that several models being used together to illustrate several possible scenarios for the future.

Bass Diffusion Model

Ohba (1984) was perhaps the first that proposed what many researchers call a flexible SRGM. This model has been developed under the assumption that the more that errors are detected, the more undetected errors become detectable (notice the similarity with Bass model assumption). The model has the following differential equation:

$$\frac{dN(t)}{dt} = b(t)[m - N(t)], \text{ where: } b(t) = b \times K(t) \text{ and } K(t) = r + (1-r)\frac{N(t)}{m} \qquad (1)$$

initial condition $N(0) = 0$.
Bittanti et.al. (1988) proposed very similar model with slight twist. Their differential equation is:

$$\frac{dN(t)}{dt} = b(m)[m - N(t)], \text{ where: } b(m) = k_i + (k_f - k_i)\frac{N(t)}{m} \qquad (2)$$

Here, k_i and k_f are initial and final values of the defect exposure coefficient. If $k_i = k_f$, then it reduces to the exponential model. If $k_f \gg k_i$, the defect growth curve become an S-shape function.
Later, Kapur & Garg (1992) assumed that the detection of errors also results in detection of some of the remaining errors without these errors causing any failure. This leads to the following differential equation (which is exactly the Bass Diffusion model):

$$\frac{dN(t)}{dt} = p[m - N(t)] + \frac{q}{m}N(t)[m - N(t)] = \left[p + \frac{q}{m}N(t)\right][m - N(t)] \qquad (3)$$

With $N(0) = 0$, the solution to the above differential equation in (3) is given by the following equation:

$$N(t) = m\frac{1 - e^{-(p+q)t}}{1 + \frac{q}{p}e^{-(p+q)t}} \qquad (4)$$

We use the term Bass model from (4) as one of our models in this study.

Gompertz Diffusion Model in SRGM

It seems that Gompertz diffusion model is very popular in Japan – Satoh (2000) stated that many Japanese computer manufacturers and software houses have applied the Gompertz curve model.

He also provided a discrete Gompertz equation and argued that it is more stabled than the continuous one. Several other papers we found, e.g., Ohishi et.al. (2005), Ohishi et.al. (2009), Prasad et.al. (2016), and Yahoobi (2021) seem to support the popularity of Gompertz diffusion model, not just in Japan, but also worldwide.
There are a number of parameterization of Gompertz. Some of those are more useful compared to others, due to how easy the parameters to be interpreted. This paper will use this reparameterization:

$$N(t) = m \times exp(- b \times c^t) \qquad (5)$$

where:

$N(t)$ =	the expected value of total defect as a function of time
m =	upper asymptote (in SRGM, m will be the total number of potential defects in the system)
b =	integration constant
c =	growth-rate coefficient.

Logistic Diffusion Model in SRGM

The Logistic diffusion model is an extended version of Malthus' simple population growth model, which stated that the rate of population growth is proportional to the population at time t. The newly extended function added a carrying capacity m, which results in a bounded population. In the case of software testing, N(t) is the number of defects found in regards to time t. The Logistics Equation can be written as the following,

$$N(t) = \frac{m}{1+\left(\frac{m-N_0}{N_0}\right)e^{-rt}} \qquad (6)$$

where N0 is the number of defects found at the theoretical time t = 0, with m being the carrying capacity, or in other words the maximum possible defect found at a given time t. The value r represents the growth rate of the defects that would occur, assuming that the population size of the defect within the system can grow up to an infinite size.
Ohishi (2009) stated that Sakata (1974) is perhaps the first that applies both Gompertz and Logistic diffusion models in SRGM in Japan. Several other papers related to Logistic Diffusion model such as: Huang et.al. (1997), Satoh & Yamada (2002), Pham (2005), Rafi et.al. (2010), Zang & Pi (2018), and finally Haque & Ahmad (2021) clearly demonstrate that this model is widely researched and used.

Stopping Rule in Testing

Dalal & Mallows (1988) is perhaps one of the most cited articles when it comes to stopping rule in software testing. They presented a very elegant mathematical model as a sequential decision problem, where an optimal stopping rule has to be found minimizing expected loss. However, Höhle (2016) wrote a blog and discuss the criticisms to Dalal & Mallows' proposal. In addition, having a ratio $\frac{c}{f}$ (where c = the cost of fixing a bug and f = the cost of testing plus the opportunity cost of not releasing the software up to time t) is actually not that easy in practice. In this paper, we propose a much simpler approach that does not depend on any cost estimation.

METHODOLOGY

As we have stated, our research objective is to help decision makers to make decision on when to stop testing, and to have a good level of confidence by understanding the testing effort.
We make the following basic assumptions for our analysis:

1. The number of defects in any (software) system is finite and unknown.

2. Perfect debugging – when a defect is discovered, it can be fix without introducing another defect.

This is usually true for dashboard development (or relatively small software development). This assumption also implies that we only have one Sigmoid (or concave) curve.
Generally speaking, even though all researchers believe that the number of defects in a software system is finite, it is still very difficult to know how many defects are there. If the software is a newer version, there could be some idea from the previous system. However, this is not necessarily true for a complex system. Even for a minor release, a small modification for a code can result into a good number of defects introduced into the system. Therefore, it is very intuitive to accept that when there are only minimal amounts of testing data, the prediction of the maximum number of defects in the software system would be a wild guess at the beginning. So, we have our first hypothesis:
H1: Regardless the method (for all methods), early prediction of the estimated number of defects in the system (m) will never be good.
As corollary, since there is no easy way to know the estimated number of defects in the system at the early period of testing, we can also have the following two additional hypotheses:
H2: The estimated number of defects in the system will therefore have a high variability for all methods.
H3: The variation of estimated number of defects in the system will be high across all methods.

As the testing progresses, it will become clearer how many defects will be in the software. Therefore, it is imperative to have the following additional hypothesis (that later can be utilized as our proposal for stopping criteria):
H4: When most defects are discovered, the estimated number of defects (for all methods) will be stable over time.
H5: The variation of estimated number of defects across all methods will also become less, i.e., all (good) SRGM will converge to similar "true" numbers of maximum defects toward the end.
We decided to present and illustrate our proposal using publicly available datasets. Some of these datasets are from open-source software projects such as: Apache and Gnome. Some are commercial software (Tandem from HP, PL/I from IBM, Stratus from Cisco, etc.), while others are government/military in the US & China. Some have multiple versions, etc. (see Table 1).

Table 1. Public datasets, their statistics, and sources

Data Sets (DS)	Test Freq.	Length of Tests	Range of Defect/Test	Avg. of Defect/Time	Known Defects	Notes/References
DS01	Daily	111	0 - 49	4.33	**481**	Real-Time (Tohma et.al 1991)
DS02a Release 1	Weekly	20	0 - 16	5	**100**	Tandem, HP, commercial (Wood 1996)
DS02b Release 2	Weekly	19	14-Jan	6	**120**	Tandem, HP, commercial (Wood 1996)
DS02c Release 3	Weekly	12	12-Jan	5.08	**61**	Tandem, HP, commercial (Wood 1996)
DS02d Release 4	Weekly	19	0 - 6	2.25	**42**	Tandem, HP, commercial (Wood 1996)
DS03	Weekly	19	Feb-37	17.26	**328**	PL/I, IBM, commercial (Ohba 1984)
DS04a Release 1	Weekly	73	0 - 15	4.93	**360**	Stratus, Cisco, commercial (Mullen 1998)
DS04b Release 2	Weekly	120	0 - 9	1.67	**200**	Stratus, Cisco, commercial (Mullen 1998)
DS05	Weekly	21	0 - 18	6.48	**136**	Government/Military (Musa 1985, Musa *et.al.* 1987, Kapur & Younes 1995)
DS06	Weekly	17	21-Jan	8.47	**144**	Mid-size (Xie *et.al.* 2006)
DS07	Monthly	60	0 - 16	2.43	**146**	webERP, open source (Li & Pham 2019)
DS08	Daily	73	0 - 13	5	**367**	Government/Military (Bao *et.al.* 2000)
DS09a Apache 2.0.35	Daily	43	0 - 8	1.72	**74**	Apache, open source (Li & Yi 2016)
DS09b Apache 2.0.36	Daily	103	0 - 5	0.49	**50**	Apache, open source (Li & Yi 2016)
DS09c Apache 2.0.39	Daily	164	0 - 3	0.36	**58**	Apache, open source (Li & Yi 2016)
DS10a Gnome 2.0	Monthly	19	8-Jan	3.94	**78**	Gnome, open source (Gandhi *et.al.* 2018)
DS10b Gnome 2.2	Monthly	24	0 - 9	2.35	**54**	Gnome, open source (Gandhi *et.al.* 2018)
DS10c Gnome 2.4	Monthly	46	0 - 7	1.17	**54**	Gnome, open source (Gandhi *et.al.* 2018)

Early Period

Table 1 indicates that there are 3 types of test frequencies, namely: daily, weekly, & monthly. Given that we need to estimate 3 parameters for each model, we decided to define our methodology as follows:

- For daily DS, early prediction is defined as the first 21, 22, …, 25 days respectively, i.e., we will use defects from days 1 – 21, days 1 – 22, days 1 – 23, days 1 – 24, & days 1 – 25 to estimate the maximum number of estimated defects in the system (m);
- For weekly DS, early prediction of m is obtained using the first 5, 6, …, 9 weeks respectively, and
- For monthly DS, we use the months 1 – 6, 1 – 7, …, 1 –10 to get the early prediction of m.

Stabilization Period

Similarly, what we define as stabilization periods are as the last 3 data points, e.g.,

- For DS01 – it means we use days 1 – 109, 1 – 110, and 1 – 111 respectively.
- For DS06 (weekly) – we will use weeks 1 – 15, 1 – 16, and 1 – 17 respectively.
- For DS10a – we use months 1 – 17, 1 – 18, and 1 – 19 respectively.

It is important to explain that we purposely pick only 3 sets of data points since we believe that some systems (out of 18 that we consider) may not have too many data points, and they may not reach stability yet. Therefore, we did not pick 5 sets of data points like in the Early period.

R-Packages to Use

To obtain the parameters m, p, and q for Bass model, we use R packages: diffusion (Schaer and Kourentzes (2018)) and also DIMORA (Federico (2021)). We choose these R packages to estimate the Bass parameters since the first one (diffusion) is simply using the traditional OLS (ordinary least square) with linear approximation suggested by Bass (1969) while the second one (DIMORA) is using NLS (non-linear least square). Similarly, we use nls with SSGompertz (self-starter Gompertz) and growthrates library (Petzoldt, 2020) to calculate Gompertz' parameters: m, b, and c. Finally, to obtain the parameters m, N_0, and r for the Logistics Equation in (12), we use packages: growthcurver by Sprouffske (2020) and growthrates by Petzoldt (2020) that are available in R.

FINDINGS AND DISCUSSION

To prove our hypothesis, we run R-packages to our 18 datasets as we have described in above and we summarize the result in Figure 1.

Figure 1 shows boxplot of CVs of m (maximum potential defect) values from five early predictions (as outline in the previous section) and three stable predictions using three SRGMs (Bass, Gompertz, and Logistic). It is very easy to see that almost all values for m's are way off from the actual known defects (with few exceptions). Similarly, almost all coefficient of variations (CVs) is more than 5%, with median 32.5%, 31%, 18.5% for Bass, Gompertz and Logistic respectively. the Therefore, this demonstrates the correctness of H1 and H2. Meanwhile for stable period predictions, practically almost all CVs for m (maximum potential defects) is now less than 5% (with very few exceptions). Notice also how close the μ_m (mean of m) to the known defects. This concludes our proof for hypothesis H4 and H5.

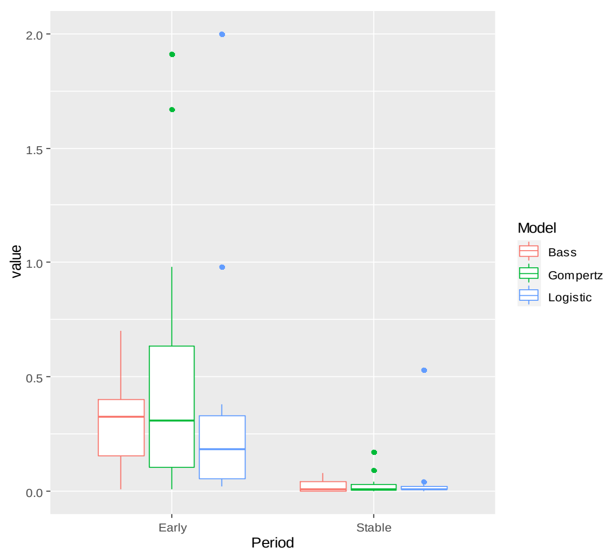

Figure 1. Coefficient of Variations from early and stable predictions using Bass, Gompertz and Logistic Models.

Table 2 shows the average and CVs of m value from three SRGMs (Bass, Gompertz, & Logistic) for the first 3 sets of early prediction data points. Again, we can easily see that most CVs tend to be more than 5% (with few exceptions). Even those few exceptions, the average value of m is clearly far from the known defect of the system. Therefore, we can conclude that this result demonstrates the correctness of hypothesis H3.

Stopping Rule Proposal

Now, we are in a good position to outline our proposal for when we can release a software (or dashboard) after a period of testing. Let's define couple more terminologies to be able to quantify our proposal clearly. Consider a distribution frequency of defects found per unit time. Based on previously stated assumption, we believe it is reasonable to assume a Bell curve. Therefore, we can have the following stable (software) system definitions as our stopping rule proposal.

Table 2. Average & CVs of m for the first 3 sets of Early Period Prediction

Data Sets (DS)	1st Early Time Period		2nd Early Time Period		3rd Early Time Period		Known Defect
	μ_m	CV(m)	μ_m	CV(m)	μ_m	CV(m)	
DS01	252.09	12.72%	253.22	8.57%	250.33	6.90%	481
DS02a Release 1	109.97	51.45%	213.99	83.92%	106.3	33.65%	100
DS02b Release 2	83.73	45.17%	109.53	53.89%	159.43	52.00%	120
DS02c Release 3	5009.96	139.95%	31091.56	141.15%	118.49	59.95%	61
DS02d Release 4	11.82	5.03%	28.48	32.15%	38.62	37.93%	42
DS03	119.32	3.29%	122.79	8.66%	244.07	54.07%	328
DS04a Release 1	18.97	4.64%	25473.97	140.26%	906.01	126.66%	360
DS04b Release 2	10.84	4.58%	15.94	6.96%	21.39	7.85%	200
DS05	7.42	45.14%	13.6	38.19%	16.27	39.55%	136
DS06	135.66	20.81%	119.26	12.70%	125.16	6.64%	144
DS07	12.5	1.13%	7877722.85	141.42%	46.27	15.98%	146
DS08	151.03	8.97%	150.83	7.98%	149.4	7.00%	367
DS09a Apache 2.0.35	110.91	54.89%	110	54.27%	113.27	53.43%	74
DS09b Apache 2.0.36	37.56	32.27%	36.97	31.73%	36.15	30.85%	50
DS09c Apache 2.0.39	48.34	46.77%	48.36	39.07%	49.54	35.47%	58
DS10a Gnome 2.0	40.15	22.21%	68.44	43.36%	84.65	24.85%	78
DS10b Gnome 2.2	55.22	37.70%	58.27	31.80%	42.71	13.61%	54
DS10c Gnome 2.4	39.63	26.95%	25.07	8.72%	28.69	9.23%	54

Definition 1: (simple, classic, & equally crucial definition)
A system under testing is called stable at time t if the moving average (MA) of defects (x) found for a certain period n, i.e., from time (t – n +1) to time t, is less than or equal to a factor (0 < α1 < 1,

usually we set α = 5% – 20%) multiply by previously known maximum defect than can be found in a unit time, i.e., we can mathematically write as:

$$\frac{x_{t-n+1}+\cdots+x_t}{n} \le \left\lceil \alpha_1 \max_t\{x_i\} \right\rceil \tag{10}$$

Please note that the larger n is, the more confidence we can have. Similarly, the smaller α1 is, the more confidence we can have that our (software) system does not have any remaining defect. We put a ceiling on the rhs of eq (10) since the number of defects should be an integer (but this is not very crucial).

Definition 2: (from H1 – H5)

A system under testing is called stable at time t if the total cumulative defects found at time t is more than or equal to a factor (β, e.g., we can set $\beta = 95\%$) multiply by the maximum potential defects (m) from our SRGM. Mathematically, we propose to represent this definition as:

$$\sum_{i=1}^{t} x_i \ge \beta f\left(m_{B_t}, m_{G_t}, m_{L_t}\right) \tag{11}$$

In the equation (11) above, $f\left(m_{B_t}, m_{G_t}, m_{L_t}\right)$ could be defined as $\frac{m_{B_t}+m_{G_t}+m_{L_t}}{3}$ (arithmetic mean) or $\sqrt[3]{m_B m_G m_L}$ (geometric mean) or even $min\{m_{B_t}, m_{G_t}, m_{L_t}\}$. The most important point, though, is that:

$$\mathrm{CV}\{m_{B_t}, m_{G_t}, m_{L_t}\} \le \alpha_2 \tag{12a}$$

Again, usually α2 = 5% is considered good for coefficient of variation. Moreover, as we have demonstrated previously, if we want to be more robust, we could also add an additional condition that $\mathrm{CV}\{m_{B_{t-n+1}}, \dots, m_{B_t}\}$ ≤ α2, $\mathrm{CV}\{m_{G_{t-n+1}}, \dots, m_{G_t}\}$ ≤ α2, and $\mathrm{CV}\{m_{L_{t-n+1}}, \dots, m_{L_t}\}$ ≤ α2, or we can simply write mathematically as:

$$max\left\{\begin{matrix}\mathrm{CV}\{m_{B_{t-n+1}}, \dots, m_{B_t}\}, \mathrm{CV}\{m_{G_{t-n+1}}, \dots, m_{G_t}\}, \\ \mathrm{CV}\{m_{L_{t-n+1}}, \dots, m_{L_t}\}\end{matrix}\right\} \le \alpha_2 \tag{12b}$$

With the above definitions, our stopping rule proposal can be illustrated by the following flowchart in Figure 2.

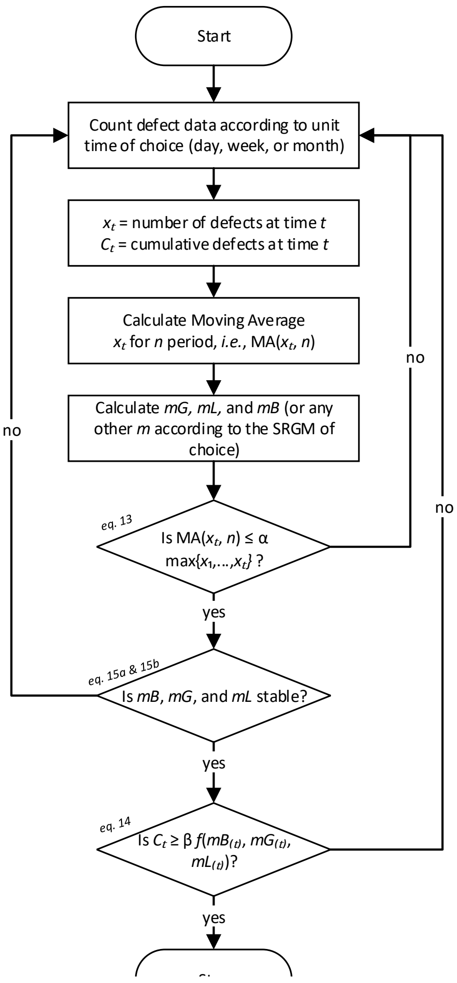
Start
Count defect data according to unit time of choice (day, week, or month)
x_t = number of defects at time t
C_t = cumulative defects at time t
Calculate Moving Average x_t for n period, *i.e.*, MA(x_t, n)
Calculate *mG, mL,* and *mB* (or any other *m* according to the SRGM of choice)
eq. 13
Is MA(x_t, n) ≤ α max{x_1,…,x_t} ?
eq. 15a & 15b
Is *mB*, *mG*, and *mL* stable?
eq. 14
Is C_t ≥ β *f*($mB_{(t)}$, $mG_{(t)}$, $mL_{(t)}$)?
yes
yes
yes
no
no
no

Figure 2. Flowchart for Stopping Rule proposal

Numerical Examples

To illustrate our stopping rule proposal with DS10b, we carried out calculation as in Table 7 (using $\alpha_1 = \alpha_2 = 5\%$, n = 5, $\beta = 95\%$, and n = 4).

Lastly and for completeness, the result of applying our proposed stopping criteria (using definition 1 only, using definition 2 only, and using definition 1 ∩ definition 2) to all 18 datasets with test frequency, $\alpha_1, \alpha_2, \beta$, and n as given here:

- Daily: $\alpha_1 = \alpha_2 = 5\%, \beta = 95\%$, and n = 5
- Weekly/Monthly: $\alpha_1 = \alpha_2 = 5\%, \beta = 95\%$, and n = 3

is summarized in Table 4.

CONCLUSION AND FURTHER RESEARCH

When the testing had discovered most defects, most (if not all) SRGMs will converge to similar value for the maximum potential defects in the system (m). Therefore, rather than contrasting various SRG models, we can utilize this information to build our confidence using several SRG models at once in order to provide stopping rule without losing too many valuable times to release the software/application that is being tested.

When the predicted values of maximum potential defects from various SRGM have small coefficient of variation, we know it very likely represents true maximum potential defects of the system. Furthermore, by requiring coefficient of variation for n consecutive periods of m to be less than α (say: 5%), we ensure that the prediction of maximum potential defects by any SRGM to be stable. Hence, increasing our confidence.

Together with a simple, practical, and yet classic definition of stable system, we can develop a robust stopping criteria for software testing. Of course, it is up to individual organization on how to implement the stopping criteria. We provided various parameters that can be tuned to suit the need.
Using publicly available datasets in various journals, we demonstrated that it is actually better to use multiple software reliability growth models (SRGMs) to build confidence, and we can identify which projects are released on a good state, and which ones are released by executive decision.

Interestingly, from all public datasets that we have collected, there are some that exhibit 2nd wave phenomenon. In SRGM, this could mean that perfect debugging assumption is not always true (the other possibility is an introduction of new feature late in the cycle). This is an interesting subject that has application in various different aspects of life and extend beyond this research.
Another area of further research is a more stable parameter prediction that do not change much with additional data. This is actually presented in Vincent et.al. (2022) already by applying similar technique as Levitt et.al. (2021), i.e., to linearize Sigmoid functions.

Table 3. Numerical Example for DS10b with α1 = α2 = 5%, β = 95%, and n = 4

Gnome 2.2				α_1 =	0.05					α_2 =	0.05	β =	0.95
week	defect	cum defect	MA(x, 4)	RunMax	def #1	m_B	CV(m_B)	m_G	CV(m_G)	m_L	CV(m_L)	CV(mB, mG, mL)	def #2
1	5	5	5	5	0								
2	4	9	4.5	5	0								
3	5	14	4.67	5	0								
4	5	19	4.75	5	0								
5	9	28	5.75	9	0	46.2		84		35.46		37.70%	0
6	5	33	6	9	0	64.45	16.49%	77.22	4.20%	33.13	3.41%	31.80%	0
7	2	35	5.25	9	0	43.77	17.93%	49.24	21.44%	35.12	2.98%	13.61%	0
8	1	36	4.25	9	0	39.67	19.55%	42.93	27.74%	37.15	4.06%	5.93%	0
9	2	38	2.5	9	0	43.7	20.25%	41.84	27.22%	38.95	6.04%	4.71%	0
10	3	41	2	9	0	43.05	3.96%	43.05	6.58%	42.03	6.62%	1.13%	0
11	2	43	2	9	0	45.15	4.68%	45.15	2.77%	44.08	6.61%	1.13%	0
13	1	44	2	9	0	46.2	2.76%	46.2	3.88%	45.1	5.52%	1.13%	0
14	0	44	1.5	9	0	46.2	2.85%	46.2	2.85%	45.1	2.85%	1.13%	1
15	4	48	1.75	9	0	50.4	4.29%	50.4	4.29%	49.2	4.29%	1.13%	1
16	1	49	1.5	9	0	51.45	4.92%	51.45	4.92%	50.23	4.92%	1.13%	1
17	1	50	1.5	9	0	50.31	4.05%	52.5	4.77%	51.25	4.77%	1.75%	1
18	1	51	1.75	9	0	51.73	1.23%	53.55	2.26%	52.28	2.26%	1.46%	1
19	0	51	0.75	9	0	52.3	1.41%	53.55	1.65%	52.28	1.65%	1.13%	1
20	0	51	0.5	9	0	52.43	1.63%	53.55	0.85%	52.28	0.85%	1.08%	1
21	0	51	0.25	9	1	52.36	0.54%	53.55	0.00%	52.28	0.00%	1.10%	1
22	1	52	0.25	9	1	52.6	0.21%	54.6	0.84%	53.3	0.84%	1.55%	1
23	0	52	0.25	9	1	52.68	0.25%	54.6	0.97%	53.3	0.97%	1.49%	1
24	2	54	0.75	9	0	56.7	3.36%	56.7	2.09%	55.35	2.09%	1.13%	1

Table 4. When to Release Software

When to Release the Software				**When to Release the Software**			
Dataset	**def #1 only**	**def #2 only**	**def #1 ∩ def #2**	**Dataset**	**def #1 only**	**def #2 only**	**def #1 ∩ def #2**
DS01	64	26	75	DS06	N/A	11	N/A
DS02 Release 1	19	N/A	N/A	DS07	23	33	N/A
DS02 Release 2	N/A	19	N/A	DS08	N/A	N/A	N/A
DS02 Release 3	N/A	12	N/A	DS09 Apache 2.0.35	33	38	33
DS02 Release 4	N/A	17	N/A	DS09 Apache 2.0.36	23	81	81
DS03	N/A	N/A	N/A	DS09 Apache 2.0.39	48	72	75
DS04 Release 1	60	62	N/A	DS10 Gnome 2.0	N/A	N/A	N/A
DS04 Release 2	50	39	50	DS10 Gnome 2.2	20	14	20
DS05	N/A	N/A	N/A	DS10 Gnome 2.4	18	18	18

REFERENCES

Cusick, J.J., 2019. The first 50 years of software reliability engineering: A history of SRE with first person accounts. arXiv preprint arXiv:1902.06140.
Wood, A., 1996. Predicting software reliability. Computer, 29(11), pp.69-77.
Chakravarty, S., 2007. Adapting Bass-Niu model for product diffusion to software reliability. Reliability: Theory & Applications, 2(1 (5)), pp.22-33.
Rafi, S.M. and Akthar, S., 2011. Software Reliability Growth Model with Bass Diffusion Test-Effort Function and Analysis of Software Release Policy. International Journal of Computer Theory and Engineering, 3(5), p.671.
Gandhi, N., Sharma, H., Aggarwal, A.G. and Tandon, A., 2019. Reliability growth modeling for oss: a method combining the bass model and imperfect debugging. In Smart Innovations in Communication and Computational Sciences (pp. 23-34). Springer, Singapore.
Yaghoobi, T., 2020. A software reliability growth model with Gompertz-logarithmic failure time distribution. International Journal of Quality & Reliability Management.
Asraful Haque, M. and Ahmad, N., 2021. A logistic growth model for software reliability estimation considering uncertain factors. International Journal of Reliability, Quality and Safety Engineering, 28(05), p.2150032.
Spiess, A.N. and Neumeyer, N., 2010. An evaluation of R2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach. BMC pharmacology, 10(1), pp.1-11.
Ohba, M., 1984. Software reliability analysis models. IBM Journal of research and Development, 28(4), pp.428-443.
Bittanti, S., Bolzern, P., Pedrotti, E., Pozzi, M. and Scattolini, R., 1988. A flexible modelling approach for software reliability growth. In Software reliability modelling and identification (pp. 101-140). Springer, Berlin, Heidelberg.
Kapur, P.K. and Garg, R.B., 1992. A software reliability growth model for an error-removal phenomenon. Software Engineering Journal, 7(4), pp.291-294.
Satoh, D., 2000. A discrete Gompertz equation and a software reliability growth model. IEICE TRANSACTIONS on Information and Systems, 83(7), pp.1508-1513.
Ohishi, K., Okamura, H. and Dohi, T., 2005, July. Gompertz software reliability model and its application. In 29th Annual International Computer Software and Applications Conference (COMPSAC'05) (Vol. 1, pp. 405-410). IEEE.
Ohishi, K., Okamura, H. and Dohi, T., 2009. Gompertz software reliability model: Estimation algorithm and empirical validation. Journal of Systems and software, 82(3), pp.535-543.
Prasad, R.S., Narayana, V.S. and Mohan, G.K., Software reliability using SPC: Gompertz.
Huang, C.Y., Kuo, S.Y. and Chen, Y., 1997, November. Analysis of a software reliability growth model with logistic testing-effort function. In Proceedings The Eighth International Symposium on Software Reliability Engineering (pp. 378-388). IEEE.
Satoh, D. and Yamada, S., 2002. Parameter estimation of discrete logistic curve models for software reliability assessment. Japan Journal of Industrial and Applied Mathematics, 19(1), pp.39-53.
Pham, H., 2005. A generalized logistic software reliability growth model. Opsearch, 42(4), pp.322-331.
Rafi, S.M. and Akthar, S., 2010. Software reliability growth model with logistic-exponential testing effort function and analysis of software release policy. In Proceedings of international conference on advances in computer science.

Zang, S. and Pi, D., 2018. Software Reliability Growth Model for Imperfect Debugging Process Considering Testing-Effort and Testing Coverage. Transactions of Nanjing University of Aeronautics and Astronautics, 35(3), pp.455-463.
Dalal, S.R. and Mallows, C.L., 1988. When should one stop testing software?. Journal of the American Statistical Association, 83(403), pp.872-879.
Zanghi Federico, 2021. DIMORA: Diffusion Models R Analysis, R package version 0.2.0, https://CRAN.R-project.org/package=DIMORA.
Kathleen Sprouffske, 2020. growthcurver: Simple Metrics to Summarize Growth Curves, R package version 0.3.1, https://CRAN.R-project.org/package=growthcurver.
Thomas Petzoldt, 2020. growthrates: Estimate Growth Rates from Experimental Data, R package version 0.8.2, https://CRAN.R-project.org/package=growthrates.
Levitt, M., Scaiewicz, A. and Zonta, F., 2020. Predicting the trajectory of any COVID19 epidemic from the best straight line. medRxiv. Preprint posted online June, 30.