なぜバグ曲線は収束するのか
～Microsoft Excel を使って考えてみる～

丹羽　岳雄†

†株式会社　日本総合研究所　開発推進部門　品質開発部
141-0022 東京都品川区東五反田 2-18-1
E-mail:　†niwa.takeo@jri.co.jp

あらまし　バグ曲線が収束する理由についてはあまり議論されることがない．この収束理由につき，同一のテスト項目が複数のテストケース中でテストされることによるものと，バグをより早く発見したいというテスト実施者の意図によるものの 2 つを提案する．この 2 つによるバグ曲線の収束は，それぞれ Microsoft Excel を用いたシミュレーションで検証することができる．

キーワード　バグ曲線，信頼度成長曲線，SRGM，テストケース，重複，収束

The Reason Why Bug Curves Level Off

Takeo NIWA†

† Quality Development Department, The Japan Research Institute, Limited
2-18-1 Higashi-Gotanda, Shinagawa-ku, Tokyo, 141-0022 Japan
E-mail:　†niwa.takeo@jri.co.jp

Abstract　The reason why bug curves level off is not discussed so much. Two reasons for this leveling off are proposed in this paper. One is a redundancy of test cases to find one bug. Another is an arbitrariness of test engineers who are willing to find bugs quickly. The leveling-offs of bug curves caused by these two reasons are able to be confirmed by simulation on Microsoft Excel.

Keyword　bug curve, Software Reliability Growth Model, SRGM, test cases, redundancy, convergence

1. はじめに

バグ曲線を活用している組織は多い．

バグが十分摘出されれば曲線が寝るという明快なロジックが直感的に理解しやすいことがその理由であろう．

しかしバグが十分摘出されれば曲線が寝ることについて，本曲線使用者の理解が直感を離れた程度にまで至っているのだろうか．啓蒙書においてもバグ曲線は寝る理由なしに説明されることが多い[1][2].

では，バグ曲線が寝るという事象は対象となるソフトウェアに対してテストがどのように行われているから生じる事象なのだろうか．バグの発見はテストが進むにつれ難しくなるからと言われることもあるが，なぜ難しくなるのであろうか．

バグ曲線については，現実のテスト結果から描かれる曲線に対し，より合致するような新しい信頼度成長モデルの構築や，反対に現実のテスト結果から描かれる曲線に合致する既存モデルの選択方法について議論されることは多い．しかしなぜ曲線が収束するのかについての議論はあまり表に出ていない．

本稿では，バグ曲線が収束していく理由について，ソフトウェアとテストケースの成り立ち上必然的に生ずるものと，テストの進め方によって生ずるものの 2 面から考察し，この考察を基に，テストを続けることによって収束していくバグ曲線の意味を考えたい．

また，考察においてシミュレーションを行うが，その際には Microsoft Office Excel を用いて考えることとする．

2. 収束理由の検討

2.1. テストケース実施順とバグ曲線

テスト終盤になるとバグの発見が少なくなるということはどういうことなのだろうか．

まず，ソフトウェアとテストケースの成り立ちから必然的に生ずるものについて考えてみる．

個々のテストケース実施の際には，テスト対象自体に内包する要因によって，意図しない項目についてもテストしてしまうことが多い．つまり項目 A について確認するために作成したテストケースを実行する場合，ごく限られたコンポーネントテスト以外のテストでは，

項目 A のみをテストすることは難しく，それに付随して別な項目 B およびその他の項目もテストしてしまうことは容易に生じうる．項目 A のテスト後にテスト実行予定の項目 B にバグがある場合，項目 B を確認するためのテストケースを実施する前に，B のバグを発見できてしまうことになる．これは，ある項目についてのテストを意図したテストケースの実施によって，別な項目のバグを意図するよりも早く発見できてしまうことを意味する．

この，あるテストケースが意図しない項目についてもテストしてしまい，意図するより早くバグを発見してしまうことは，テスト終盤になるとバグの発見が少なくなるということの理由，つまりバグ曲線が収束する理由になりそうである．これは 2.2 にて検討する．

次に，バグ曲線が収束する理由として，テストの進め方によって生ずるものを考える．

テスト実施者は，バグをできるだけ早くかつできるだけ多く出したいと意図してテストを設計し実施することが多いが，これがバグ曲線になんらかの影響を及ぼすのではないだろうか．例えば複雑な機能の開発や機能の追加開発においてテストケースの実施順を考える際に，複雑な機能や追加機能をまずテストしその後それほど複雑でない機能やレグレッションテストを行う等は普通に行われることである．これはバグをできるだけ早くかつできるだけ多く出したいという意図と考えることができるし，現に複雑な機能や追加機能は経験的にもバグを発見しやすい部分である．

このような，バグをできるだけ早くできるだけ多く出したいという意図も，バグ曲線が収束することの理由になりそうである．これは 2.3 にて検討する．

ここで述べた 2 つのバグ曲線収束の理由である，ソフトウェアとテストケースの成り立ち上必然的に生ずるものとバグをできるだけ早くできるだけ多く出したいという意図について，Excel を用いたシミュレーションを行い，それが収束する理由となりうるか，また，バグ曲線の収束にどのような影響を与えているかを次節以降で考察する．

2.2. ソフトウェアとテストケースの成り立ち上必然的に生ずるもの

本節以降のシミュレーションでは，以下のようなテストケーススイートを想定する．

当該テストケーススイート中のテストケース数は N 個ある．

当該テストケーススイートで発見できるバグ数は M 個ある．

発見できる M 個のバグのそれぞれを識別するため，便宜上 1 から M の番号をつけた場合にその内の m 番のバグを発見できるテストケースの数，つまり意図せず発見してしまう重複も含んだテストケースの数を n 個とする．

なお，本稿では，テストケーススイートを前提としたテストの場合のバグ曲線について考えることの便宜上，曲線の横軸を実施済テストケース数とする．

さて，前節においてソフトウェアとテストケースの成り立ち上必然的に生ずる成長曲線の収束理由の候補として，テスト項目の重複があることが分かった．本節ではテスト項目の重複が本当にバグ曲線の収束に寄与するかを検討する．

なお，テスト実施者の意図に由来するものは，2.3 と 2.4 で考察するので，ここではテストの実施順は実施者の意図によらないものとする．この場合，テスト実施順は全てのテストケーススイートからランダムに実施しても，収束するに十分なテストケース数を持ったテストケーススイートであれば，出来上がるバグ曲線は同じ曲線になるはずである．

以上を踏まえ，テストケース全てをランダムに実施した際，あるバグ(m)が最初に発見される順番（k）を Excel 関数で表すことを考える．

テストケース全てをランダムに実施した際，バグ(m)を発見できるあるテストケースの実施順番は，テストケーススイートの 1 番目から最後の N 番目のテストケースのどこかで実施されることから，Excel 関数では，

RANDBETWEEN(1,N)　　（1）

と表すことができる．RANDBETWEEN(i,j)は区間 i，j において一様に分布する整数の乱数を返す Excel 関数である．

よって，バグ(m)を発見できるテストケース数(n)が複数ある場合，バグ(m)が最初に発見される順番（k）は，

MIN(RANDBETWEEN(1,N),RANDBETWEEN(1,N),・・・RANDBETWEEN(1,N))　　（2）

と表すことができる．MIN(a,b)は()内の引数のうち，最小の数値を返す Excel 関数である．

MIN 関数の中の RANDBETWEEN(1,N)の数は同じバグを発見できるテストケースの重複の数(n)である．

但し，式（2）におけるそれぞれの RANDBETWEEN(i,j)の内，同じ値となるものがある場合，テスト項目の重複数が小さくなる．例えば重複の数(n)が 4 の場合に RANDBETWEEN(i,j)で得られた 4 つの数の組が 7110，240，8721，240 であった場合，240 が同じ値なので，実際の重複の数は 3 となる．よって式(2)では，重複の数はこの場合，正しくは 4 ではなく 4 以下となるが，本稿では式(2)を用いてシミュレーションすることとする．

以上から，最初に発見される順番（k）は，バグ(m)を発見できるテストケースの重複の数(n)が大きくな

るほど，小さな値になる可能性，つまりテストの早い段階で発見される可能性が高くなることがわかる．

数式(2)で定めた，このバグ（m）が最初に発見される順番（k）の算出を，当該テストケーススイートで発見できる全てのバグ（m=１～M）に対して行った上で，横軸に実施したテストケース数，縦軸に累積発見バグ数をとり，発見されたバグ数をテストケース数の小さい方から累積的にプロットしていけば，テスト項目の重複により生まれるシミュレーション上のバグ曲線が得られる．

以下に，全テストケース数 N=10000，発見できるバグの数 M=200, また全てのバグにおいて発見できるテストケース数 n がそれぞれ 1(重複無)，3，5 の場合について数式(2)を用いたシミュレーションの実行例を挙げる．

表 I： 各バグがただ 1 つのテストケースでのみテストされる例

テスト実施区間 (i, j)	発見障害件数		
	重複無し	3 件重複	5 件重複
1, 10000	200	0	0

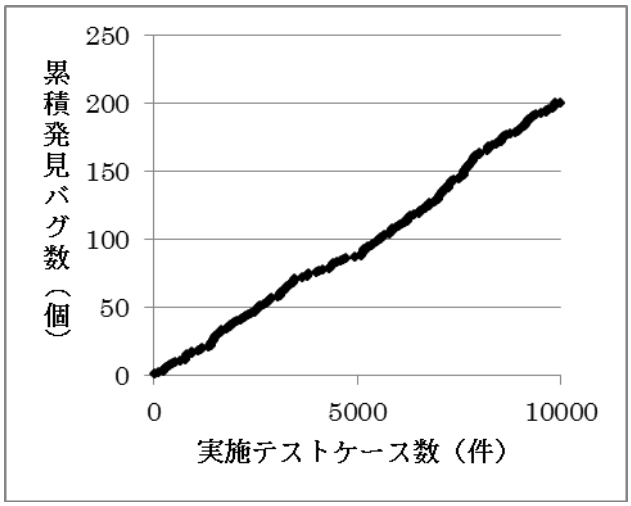

図 1： 各バグがただ 1 つのテストケースでのみテストされるバグ曲線の例

このバグ曲線は，ほぼ直線となっており，収束の傾向はみられない．

表 II: 各バグが各々3 つのテストケースでテストされる例

テスト実施区間 (i, j)	発見障害件数		
	重複無し	3 件重複	5 件重複
1, 10000	0	200	0

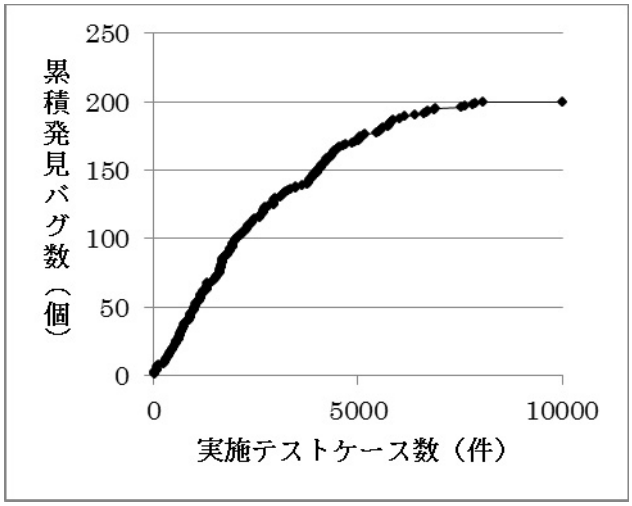

図 2： 各バグが各々3 つのテストケースでテストされるバグ曲線の例

このバグ曲線は，図 1 対比では，収束の傾向がみられる．

表 III： 各バグが各々5 つのテストケースでテストされる例

テスト実施区間 (i, j)	発見障害件数		
	重複無し	3 件重複	5 件重複
1, 10000	0	0	200

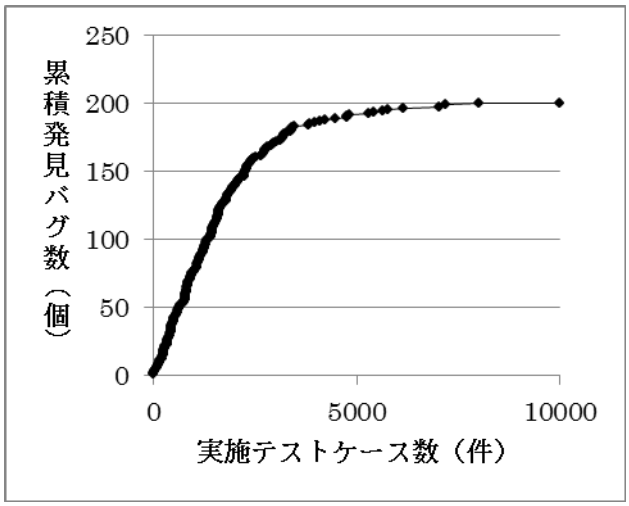

図 3： 各バグが各々5 つのテストケースでテストされるバグ曲線の例

このバグ曲線は，図 2 よりさらに早い段階での収束傾向がみられる．

これら 3 つのグラフから，各々のバグにおいて発見できるテストケースの重複数（n）が多いほど早く収束する傾向が見てとれる．

よって，このテスト項目の重複はバグ曲線を収束させる理由の 1 つと考えることができる．

2.3. テスト実施におけるテスト実施者の意図に由来するもの

本節では実施者がバグをより早く発見したいと意図することによりバグが先取りされることがバグ曲線の収束に影響を与えることについて考える．

バグを早く発見するために，難しい計算ロジックの部分，改定案件において新規に改定した部分やその部分と結合している部分との摺合せ等をテスト実施者はできるだけ早くテストしようと考える場合がある．また，テスト実施の効率を考えてテストケースの実施順を考えることにより，特定のバグが発見されるタイミングが早くなったり遅れたりという傾向を生じうる．例えば，以下のようなものがある．

①バグ原因となる箇所を予測したテスト

バグ原因となる箇所を効率よく発見するためにテスト実施者は，バグを予測し，また発見バグを基に周辺バグ，類似バグの有無を確認する等，実施内容やテスト実施順に偏りをもったテストをすることがある．この企てが機能すれば，バグは早い段階に抽出されることになる．

②恣意的な実施順の決定

テストの効率的な実施のために，テストケースをその類似性や実施容易性等を考慮してグルーピングする等，テスト実施者の一定の恣意によって実施順が決定されることがある．この恣意的に定められた実施順によってバグ発見が早くなったり遅くなったりすることが生じうる．先の①と異なるのは，テスト実施者の意図がバグ原因となる箇所を効率よく発見するためではない場合があることである．

テストケースを挙げて実行するテストの場合，テスト実施者の意図が入ることが多いので，各テストケースの実施順番がテストケーススイート全体でランダムに行われることは少なく，通常は実施順番にテストの内容からみた偏りがある．このため 2.2 で検討したソフトウェアとテストケースの成り立ち上必然的に生ずるものと合わせて，この実施順番の偏りについても，テスト終盤になるとバグの発見が少なくなることの理由として検討すべきである．

このような考えをもとに，バグ m を発見できるテストケースがテストケース順の i 番目から j 番目のテストケースの間に発見されると想定した場合，その順番（k）は，

RANDBETWEEN(i,j)　　（3）

と表すことができる．

以下に，この場合のバグ曲線のシミュレーションを行ってみる．

次のシミュレーションケースでは，テスト実施順 1 番目から 2000 番目までに発見されるバグの数が 90 個，テスト実施順 2001 番目から 4000 番目までに発見されるバグの数が 40 個，テスト実施順 4001 番目から 10000 目までに発見されるバグの数が 20 個，テスト実施順 6001 番目から 10000 目までに発見されるバグの数が 8 個としている．なお，ここでは同一バグが複数のテストケースで発見されない場合（n=1）を想定している．同一バグが複数のテストケースで発見される場合については 2.4 で考察する．

表 IV： テスト実施者の意図が影響する例

テスト実施区間 (i, j)	発見障害件数		
	重複無し	3 件重複	5 件重複
1, 2000	90	0	0
2001, 4000	40	0	0
4001,10000	20	0	0
6001,10000	8	0	0

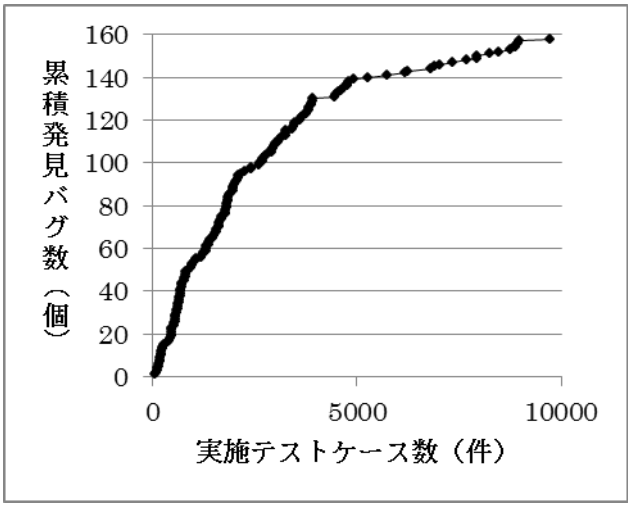

図 4： テスト実施者の意図が影響するバグ曲線の例

このバグ曲線は，収束の傾向がみられる．これは，テスト終盤になるとバグの発見が少なくなるようにシミュレーションデータを採っているので自明なことではある．

2.4. テスト項目の重複とバグの先取り後送りを考慮したバグ曲線の考察

現実のテストにおいては，テスト項目の重複とテスト実施者の意図それぞれは複合的に組み合わさってバ

グ曲線を形づくると考えられる．

この場合，バグ(m)を発見できるテストが，テストケース順の i 番目から j 番目のテストケースの間に n 個の重複したテストケース数で発見されるとすれば，このバグ(m)が最初に発見される順番(k)は，MIN(RANDBETWEEN(i,j),・・・RANDBETWEEN(i,j), RANDBETWEEN(i,j))　　（4）

と表すことができる．MIN 関数の中の RANDBETWEEN(i,j)の数は n 個である．

与えられた条件におけるバグ曲線のシミュレーション結果は，それぞれのバグ（m＝1～M）について式（4）に当てはめることで得られる．

テスト項目の重複とテスト実施者の意図の両者の複合形について検討した式（4）を基に Excel を用いたシミュレーションを行いバグ曲線の傾向を見てみる．

まず図 5 のシミュレーションでは，テスト実施の序盤（1 から 3000 番目）に全体の 30%のバグを発見し，テストケース 60%終了時点で 92%のバグを発見するような例を考える．N=10000，M=300 としている．

表 V：テスト項目の重複とテスト実施者の意図両者が複合する例①

テスト実施区間 (i, j)	発見障害件数				
	n=1	n=2	n=3	n=4	n=5
1, 3000	15	15	15	15	30
3001, 6000	0	6	6	12	12
6001,10000	8	8	8	0	0
1, 6000	30	30	30	30	30

n：重複の数

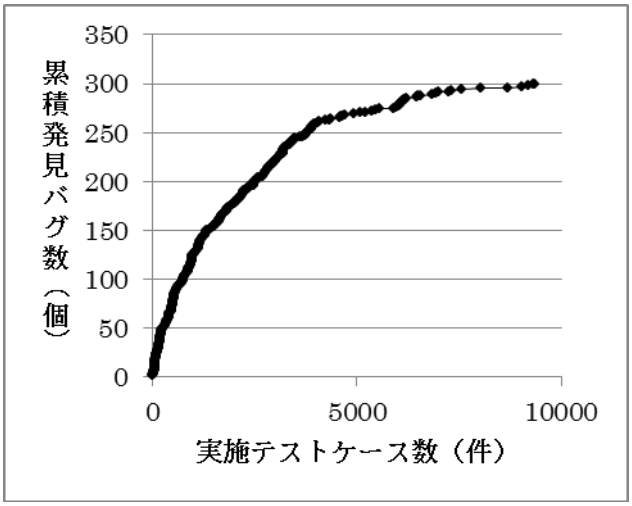

図 5：テスト項目の重複とテスト実施者の意図両者が複合するバグ曲線の例①

このバグ曲線は，テスト終盤になるとバグ発見が少なくなり，収束に近づいているようにみえる．

次に，表 V の例から(i,j)=(1,6000)の組合せ分を除いた場合を考える．

ここでは，N=10000，M=150 となる．

表 VI：テスト項目の重複とテスト実施者の意図両者が複合する例②

テスト実施区間 (i, j)	発見障害件数				
	n=1	n=2	n=3	n=4	n=5
1, 3000	15	15	15	15	30
3001, 6000	0	6	6	12	12
6001,10000	8	8	8	0	0

n：重複の数

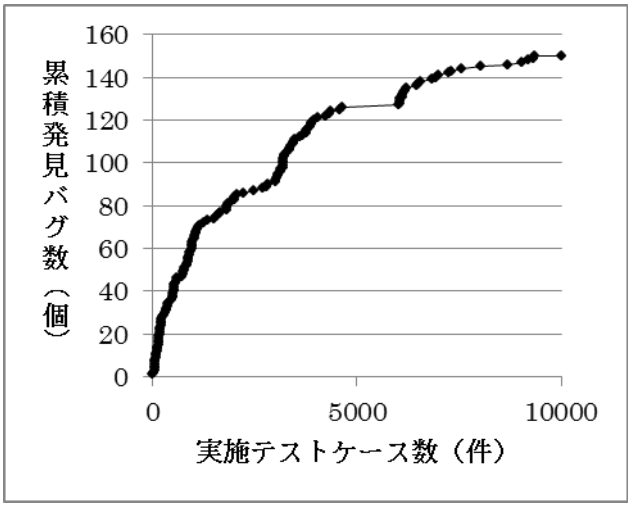

図 6：テスト項目の重複とテスト実施者の意図両者が複合するバグ曲線の例②

図 6 は図 5 対比曲線の滑らかさに欠ける．このような曲線が現実のバグ曲線において見られることは珍しくない．これは，本稿の考えではテストケースの実行順がランダムではなく，何らかのテスト実施者の意図が関与しているためバグが発見されるテストケースの実施順に偏りが生じていることによるものと考えることになる．例えば図 6 の場合，データの構成から 3 つの曲線の組み合わせととらえ，テストの序盤中盤終盤で異なった意図でのテストが行われるため，それぞれで収束傾向を示している可能性があると考えられる．

3. 結論

2．における検討により，テスト項目の重複とテスト実施者の意図が，バグ曲線の収束原因になることが分かった．

これにより，収束原因について考慮していなかったときとは異なり，これからはこれら 2 つの収束原因がバグ曲線に与える影響とその意味について考慮しながら，テストケースの作成や収束状況の確認を行うことができるようになった．

特にテスト実施者の意図によるものは，同じテストケーススイートのテストであってもテスト実施者やテスト実施環境によってテスト実施順が大きく変わり，異なるバグ曲線となる可能性があるので，過去のバグ曲線の傾向や信頼度成長曲線を用いた分析を行う際には注意が必要である．

今後の課題として，現実のテストにおける適用検証とその他の収束理由の有無調査及びそれら収束理由それぞれの収束への寄与率の検討を行うことで，テスト項目の重複とテスト実施者の意図が実際のバグ曲線にどのように影響を与えているかを検証していくべきと考える．

なお，今回は予め定めているか後付けとなるかはともかく個別の確認項目に着目したテストケースの集合体であるテストケーススイートが存在する場合のバグ曲線について検討している．しかしこのようなテストケーススイートを前提としないテストの場合，例えばいわゆるベータテストや，現実に想定されると同等の内容や量を何日分かテストするようなサイクルテストでは，そもそも同値分割の観点からは同じテストを繰り返し実施することが暗黙の前提となっていること，および，特にテストを進めていく上で，確認項目の観点におけるテスト実施者の意図はないことが多いので，本稿での検討とは異なる結果となる可能性があり，別途考察すべきものと考える．

文　献

[1] 情報処理推進機構ソフトウェアエンジニアリングセンター(編)，定量的品質予測のススメ，オーム社，2008

[2] 情報処理推進機構ソフトウェアエンジニアリングセンター(編)，続 定量的品質予測のススメ，佐伯印刷，2011

[3] Stephen H. Kan，Metrics and Models in Software Quality Engineering (2nd Edition) ，Addison-Wesley Professional，2002

[4] Stephen H. Kan，ソフトウェア品質工学の尺度とモデル，(株)構造計画研究所，2004，[3]の日本語版

[5] 松尾谷 徹，“ソフトウェア構造を考慮した信頼度モデル”電子情報通信学会技術研究報告．R2000-10, pp.7-12, Jun.2000.