White Paper, DRAFT

Software Reliability Models

Main question for Reliability Software Analysis is following – "**Is Software ready for release**?" This analysis may be performed at the expected time of Finish of Testing and at the Intermediate point of testing. If answer for this question will be negative, it is useful to predict expected time of testing finish to satisfy required value of "Bug Amount" after testing finish, at the field. So, second question may be "**When Software will be ready for release**?"

Two types of analysis can be used:

- Qualitative Analysis, based on non-parametric methods of input statistics handling
- Quantitative Analysis, based on building of some parametric models and definition of its parameters

For Software Reliability analysis usually the following time-series are used:

- "Number of days from start of testing" or "Cumulative Effort of testers from start of testing" (CE), as input parameters on the models
- "Cumulative Amount of Bugs" (CB), as output parameter

Qualitative Analysis can answer only for first question. By means simple handling of input statistics we can build curve of "Cumulative Amount of Bugs depending of Time or Cumulative Effort" and after this to analyze " is this curve has a Plateau near analyzed point T"? Fig. 1 presents typical curves of non-parametric analysis.

Main drawbacks of such analysis are following:

- This analysis is too subjective. We don't use not some formal (numerical) criteria to evaluate accuracy, confidence and other parameters of our answer (is it Plateau or not), so different experts may get different answers.
- We could not predict behavior of testing for future, so we could not answer (if it us necessary!) for second question - "**When Software will be ready for release**?"

So, we will use mostly **Quantitative Analysis.**

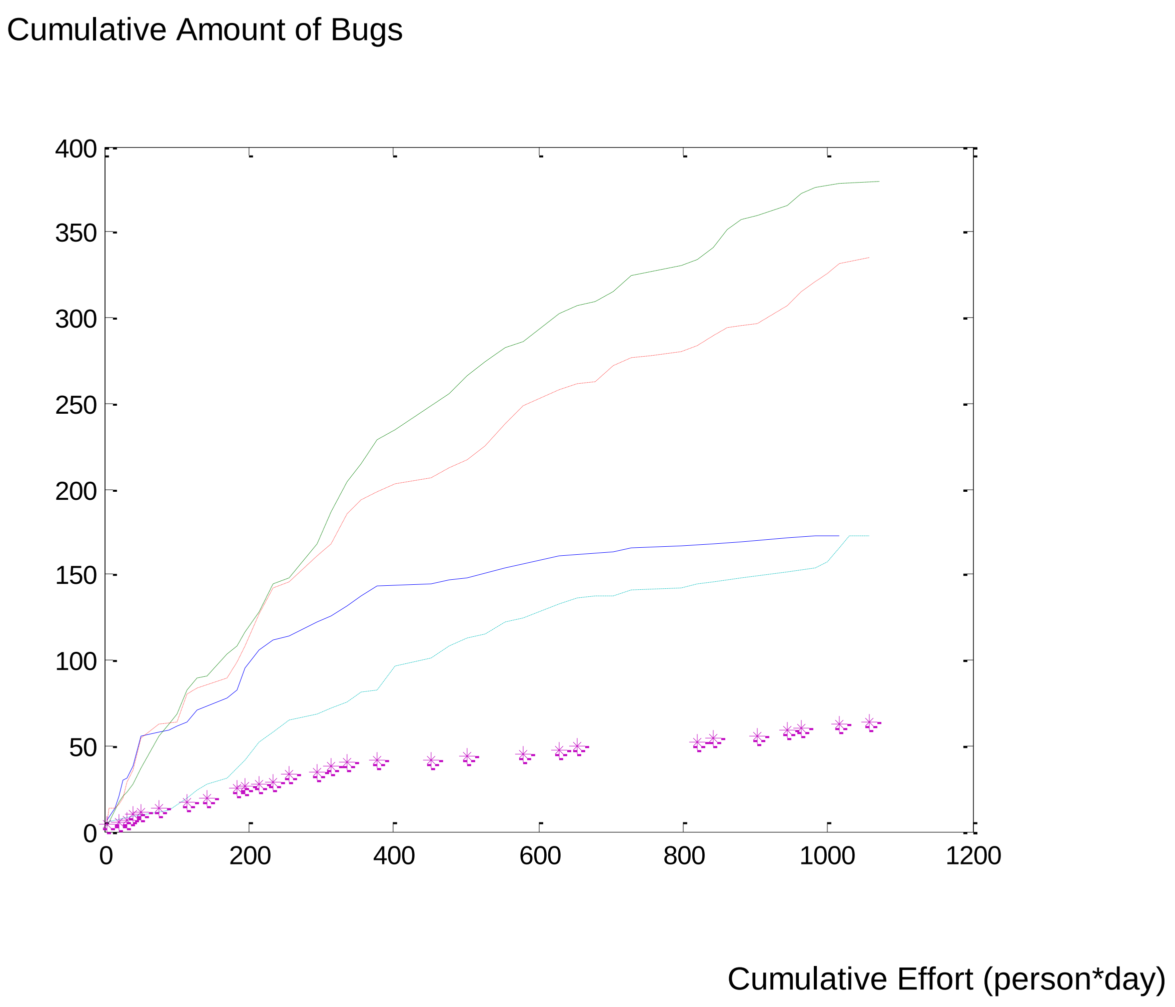


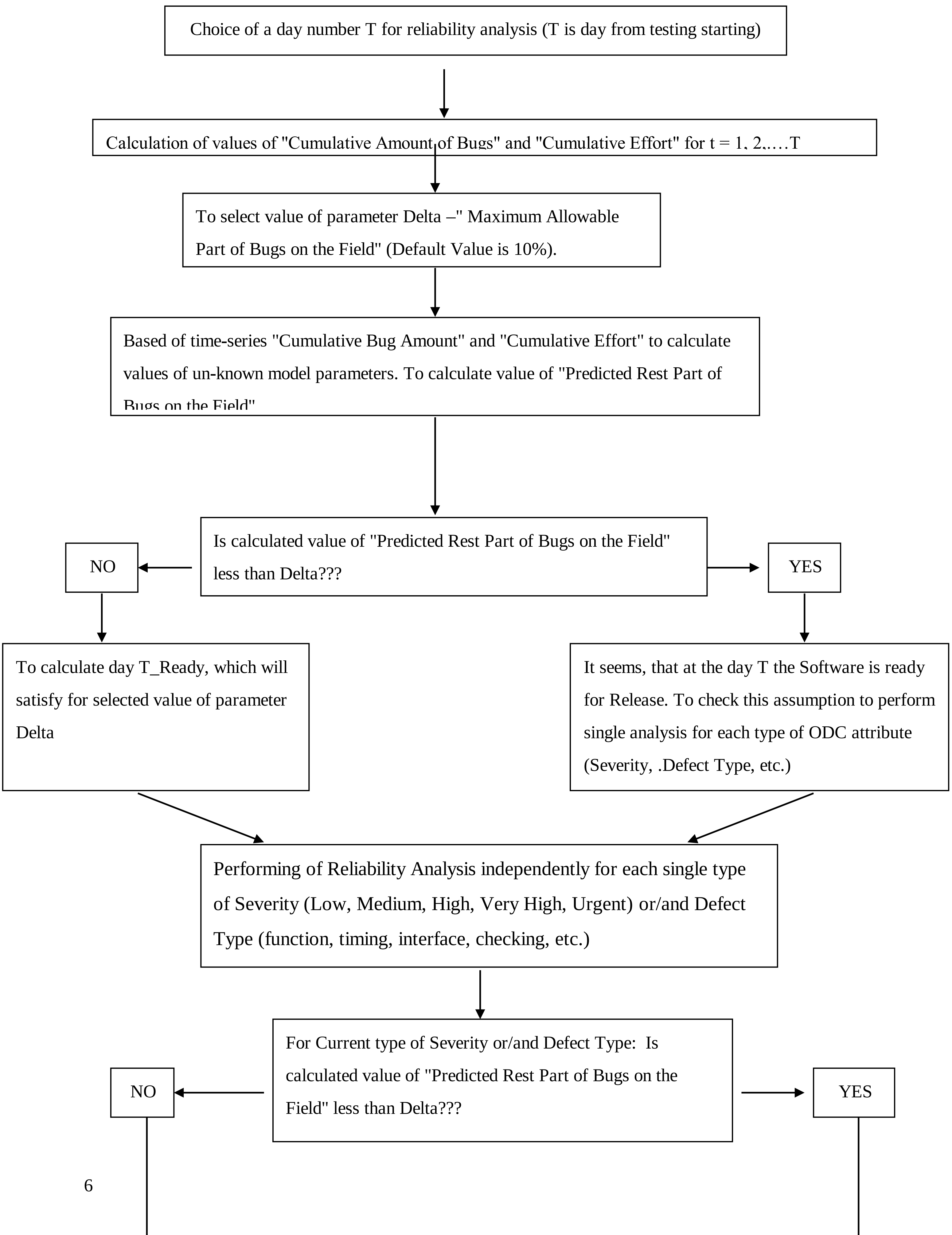
Figure 1.

The parametric model approach, one commonly associated with reliability, is the evaluation of the number of errors (failures, bugs) in the code. From an bugs trending perspective, our model employs a straightforward implementation of the some General Model to compute a nonlinear approximation to the cumulative date found to date. Rather than using the probabilistic approach to determine model parameters, however, we apply an "iterative" process that employs a non-linear optimization technique to fit the model-generated cumulative error distribution curve to the actual cumulative error distribution curve. A sum-of-difference-squared is used to determine the "best fit." Using the computed values and other functions defined by model allow us to estimate: (a) the number of errors remaining in the product, and (b) the time needed to detect the remaining errors.

Parametric modeling provides an easy way to monitor trends, but is not capable of suggesting corrective actions due to the inadequate capture of the semantics behind the defects. Orthogonal Defect Classification (ODC) is a scheme to capture the semantics of each software defect quickly. It is the definition and capture of defect attributes that make mathematical analysis and modeling possible. Is tested system ready for release? A typical parametric model can say "Yes". But when we classified the defect data using ODC, just the analysis by Defect Type may portray a different picture..

So, we suggest to perform Parametric Analysis not only for "ALL BUGS", but also according different ODC attributes – Severity, Defect Type, etc.

Flow Chart of Software Reliability Analysis



Choice of a day number T for reliability analysis (T is day from testing starting)

Calculation of values of "Cumulative Amount of Bugs" and "Cumulative Effort" for t = 1, 2,….T

To select value of parameter Delta –" Maximum Allowable Part of Bugs on the Field" (Default Value is 10%).

Based of time-series "Cumulative Bug Amount" and "Cumulative Effort" to calculate values of un-known model parameters. To calculate value of "Predicted Rest Part of Bugs on the Field"

Is calculated value of "Predicted Rest Part of Bugs on the Field" less than Delta???

To calculate day T_Ready, which will satisfy for selected value of parameter Delta

It seems, that at the day T the Software is ready for Release. To check this assumption to perform single analysis for each type of ODC attribute (Severity, .Defect Type, etc.)

Performing of Reliability Analysis independently for each single type of Severity (Low, Medium, High, Very High, Urgent) or/and Defect Type (function, timing, interface, checking, etc.)

For Current type of Severity or/and Defect Type: Is calculated value of "Predicted Rest Part of Bugs on the Field" less than Delta???

For this type of Severity/Defect point of view the Software isn't Ready for Release. In future to take attention for this type of Severity/Defect

For this type of Severity/Defect point of view the Software is Ready for Release

Consider single aspects of this process more detaily.

1. **Cumulative Effort as Main Input Indicator**

For Software Reliability analysis usually the following time-series are used:

- Number of days from start of testing, as Input
- Cumulative Amount of Bugs, as Output

Using of first parameter is possible only under assumption, that amount of testers approximately don't change during testing process. But we see from real data, that amount of testers is changed essentially during single project testing. Hence we propose to use another parameter instead of "Number of days", we will use parameter "Cumulative Effort of testers from start of testing" and will measure it in units "person*day"

To calculate value of parameter "Cumulative Amount of Bugs" depending of "Date Number" is simplest task – we only have to take into accounts ALL bugs before required "Date Number" (from analyzed project).

To calculate value of parameter "Cumulative Effort" depending of "Date Number" isn't too easy, because from input statistics we usually don't know, what tester worked at single day for this project and what tester didn't work. To calculate "Cumulative Effort" approximately, we assume, that for each single project we can to use following CONTROL PARAMETER:

- TP (Time Pause) - Significant Value of Non-Working Time

We assume, that if single tester during some time period didn't insert records about bugs more than this time, he/she didn't work on this project during this period. Assume, that TP = 7 days. If single tester didn't record bugs during period of 5 days, we will take into account its effort (5 person*day) for Cumulative Effort calculation. We assume, that tester really worked for this project, but didn't record bugs on EACH day, rather he/she recorded several bugs (sometimes 15…20) on one day on week (it is typical situation of real statistics!). But if single tester didn't record bugs during period 3 weeks (more than TP selected value = 7 days), we assume, that he/she didn't work on this project these 3 weeks.

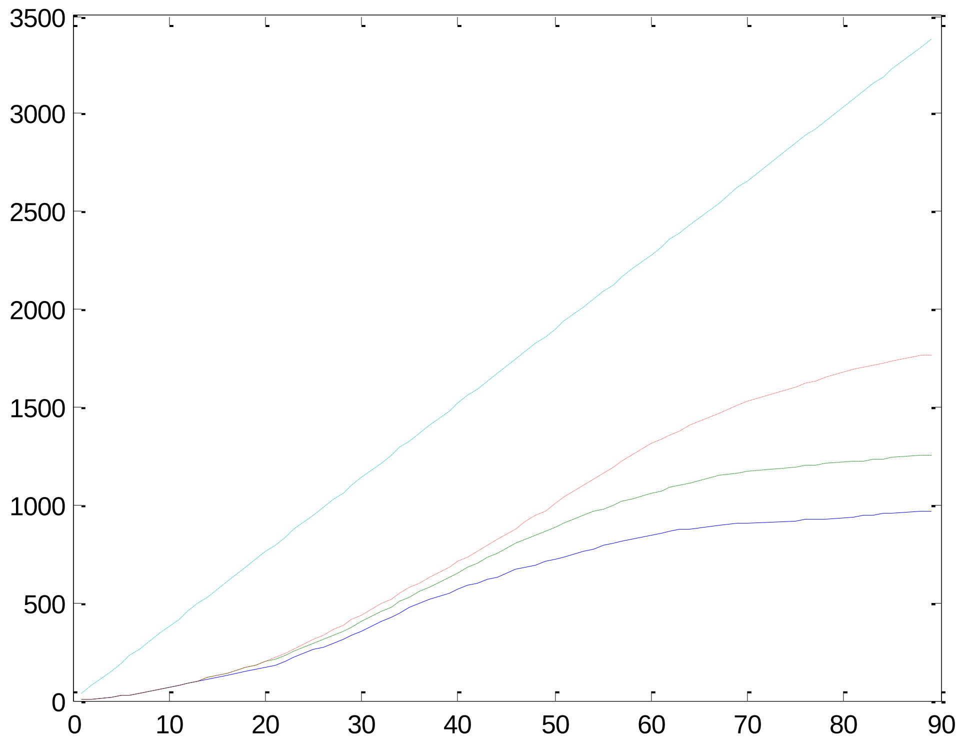
Certainly, this assumption is very approximate. Really for one tester we have to use TP = 5 days and for other tester TP = 12 days, on one situation tester has recorded 25 bugs per day after pause of 25 days (!) and on other situation he recorded 5 bugs after pause of 10 days, sometimes single tester can simultaneously work for several projects… But really it impossible to take into account ALL these factors, so we will choice TP value for full single project.

Using this approach, we can build Cumulative Effort for each single tester of analyzed project and after this to get Cumulative Effort (depending of Day number) for full project.

Value of TP essentially influences for changing of parameter "Cumulative Effort" depending on "Date Number". For example, for we will get following curves (see fig. 2):

- --------- for TP = 7 days
- - - - - - - for TP = 14 days
- ……… for TP = 28 days
- -.-.-.-.- without TP using (TP = infinite)

Cumulative Effort (person*day)



Number of Day from Testing Start

Figure 2.

For default we use TP = 7 days, but for each single investigation we select optimal value of TP (as all other control parameters) by means of minimization of sum-of-difference-squared between model-generated cumulative bug distribution curve to the actual cumulative bug distribution curve.

2. **Parametric Models for Bug Amount Prediction**

Generally speaking, we can divide parametric models into two classes:

- Models, that assume **infinite** number of failures, that can be experienced in infinite time
- Models, that assume **finite** number of failures, that can be experienced in infinite time

Typical example of first class model is Duane Reliability Growth Model:

- CB = Lambda*(CE^Beta); Rate = Lambda*Beta*(CE^(Beta-1)),

Where:
CE is Cumulative Effort (input parameter, i.e. variable)
CB is Cumulative Amount of Bugs,
Rate is Bug Rate,
Labbda and Beta are unknown control parameters, searched by means of optimization of Input Statistics.

Typical examples of second class models are following:

- Goel Model:
  CB = N*(1-exp(-b*CE)); Rate = N*b*exp(-b*CE);

- Weibull Model:
  CB = N*(1-exp(-b*(CE^a))); Rate = N*a*(b^(a-1))*exp(-b*(CE^a));

- Classical S-shaped Model:
  CB = N*(1-(1 + b*CE)*exp(-b*CE)); Rate = N*(b^2)*CE*exp(-b*CE);

- **Ohba S-shaped Model:**
  CB = N*((1-exp(-b*CE))/(1+C*exp(-b*CE))),

These models use following definitions:
N is the expected total amount of bugs to be eventually detected;
N, a, b, C are unknown control parameters.

Drawbacks of the first class models are following:

- They don’t allow us to predict rest of the total amount of bugs after testing finishing (i.e., on the field)
- Bug Rate reduction is too slow.

For some software projects the best model may be, e.g., Goel model, and for other – S-shaped model. We have compared using of different models for some real project.

Results of model comparison are shown on fig.3, where:

----- is curve by input statistics

- - - - is curve by Ohba S-shaped model

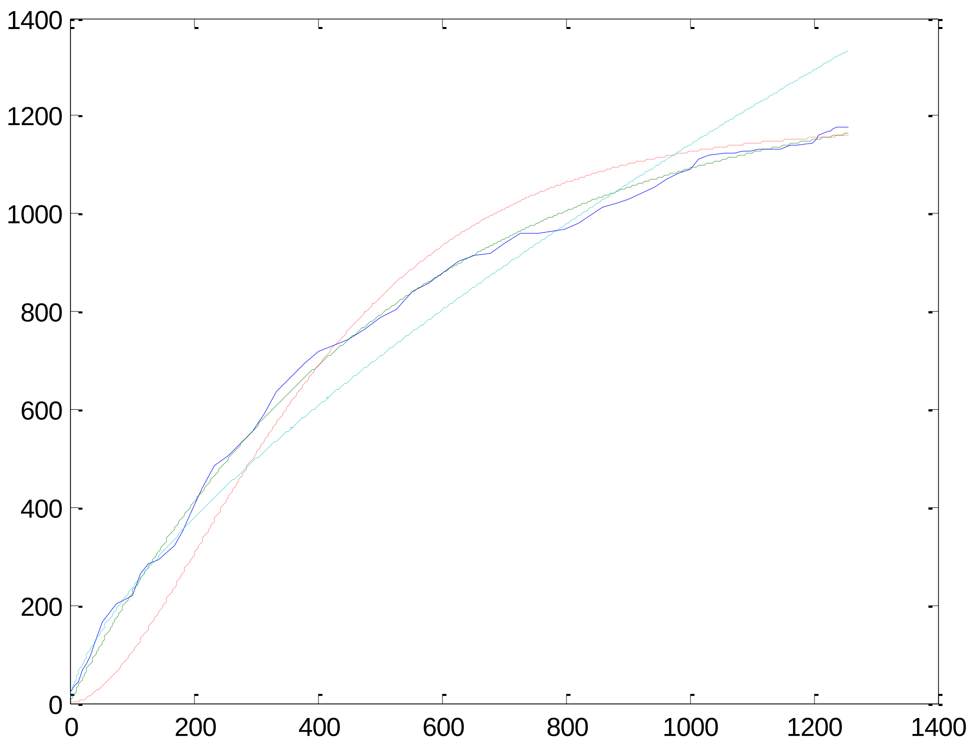
…… is curve by classical S-shaped model

-.-.-.-. is curve by Duane model

For our point of view, Ohba model is most suitable for Software Reliability estimation. But even this, most flexible model, sometimes works very bad, because it does not take into account some specific features of software testing process:

- Rate of bugs discovery sometimes essentially depends of experience of tester – with experience increasing simultaneously rate of bugs discovery will be increased
- Rate of bugs discovery sometimes essentially depends of CE rate of tester – with CE Rate increasing simultaneously rate of bugs discovery will be increased

Cumulative Amount of Bugs



Cumulative Effort (person*day)

Fig. 3

To take into account some detailed moments of software testing process, we have developed following Advanced Model:

- **CB = N*((1-exp(-b*CEM))/(1+C*exp(-b*CEM)))**, where

p and f are unknown control parameters,
CEM is Modified Cumulative Effort,

- **Δ CEM = Δ CE*(CE^ f)*(Mean_Deriv^p)**, **CEM(i+1) = CEM(i) + Δ CEM**, where

  Δ CE = CE(i+1) – CE(i).

Mean_Deriv is Mean Value of derivative (rate) of parameter CE depending of current time (day number),

Mean_Deriv(j) = ($\sum_{i=1}^{M}$ Deriv_CE(t – i +1))/M,

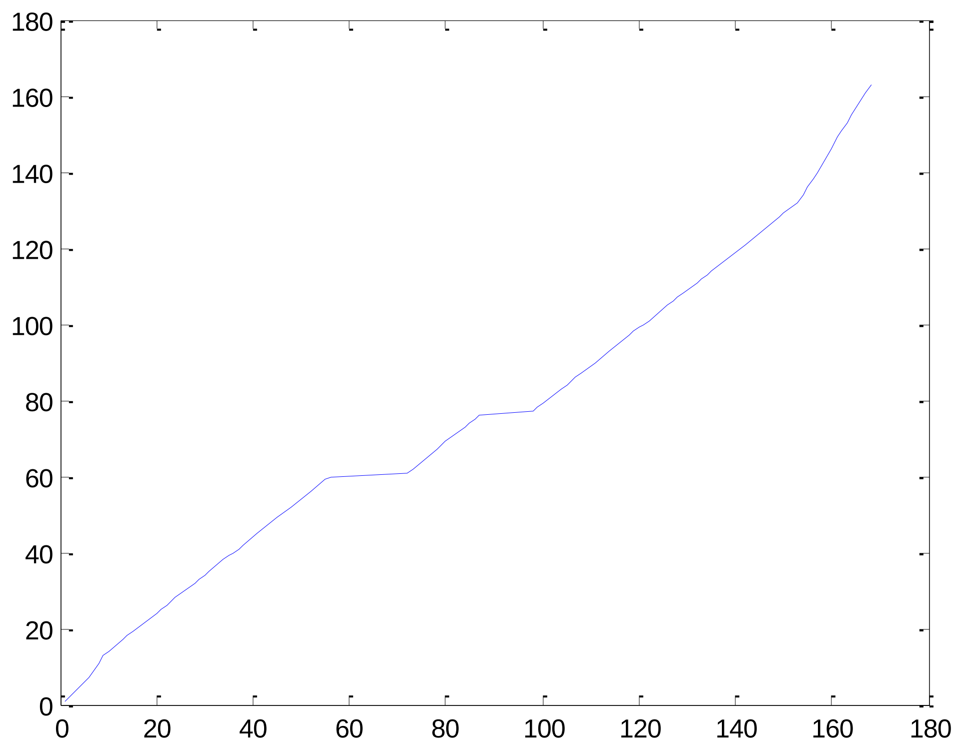
M – interval of averaging of Derivative of CE (it is also control parameter)

Deriv_CE(t) = (CE(j) – CE(j – 1))/(t(j) – t(j – 1)),

t(j) – day number at the measurement index j.

Proposed model can sense some situations, that cold not sense early developed, standard Software Reliability Models (e.g., classical S-shaped or Ohba). Consider, for example following (real !) – see Fig. 4 (for Cumulative Effort) and fig.5 (for Cumulative Bugs).

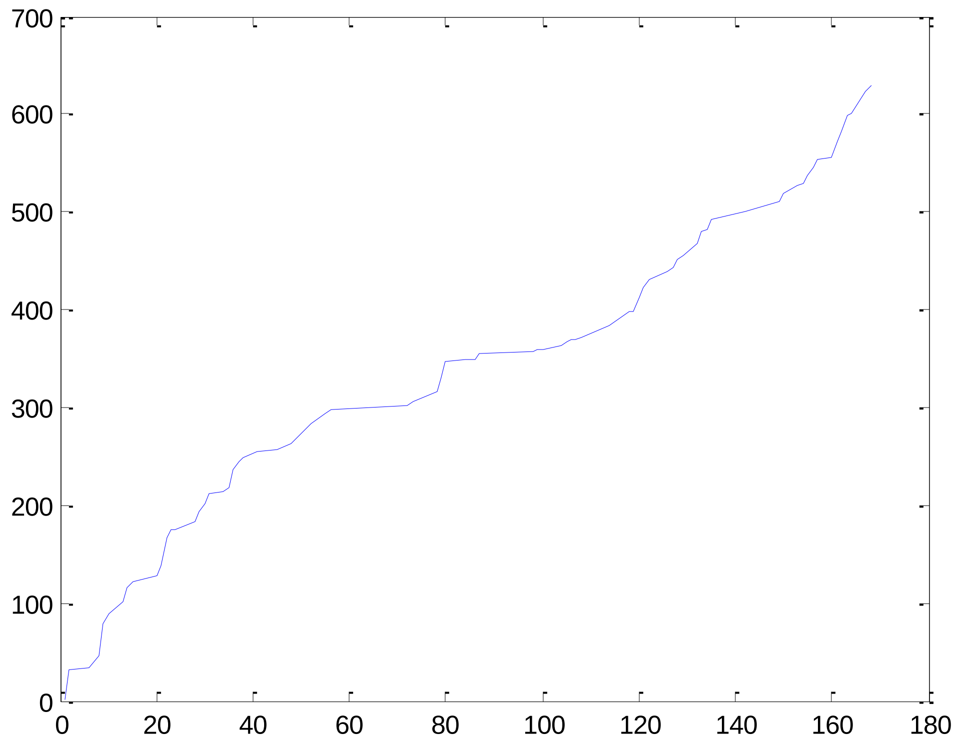
Cumulative Effort



Time (days)

Figure 4.

Cumulative Bugs



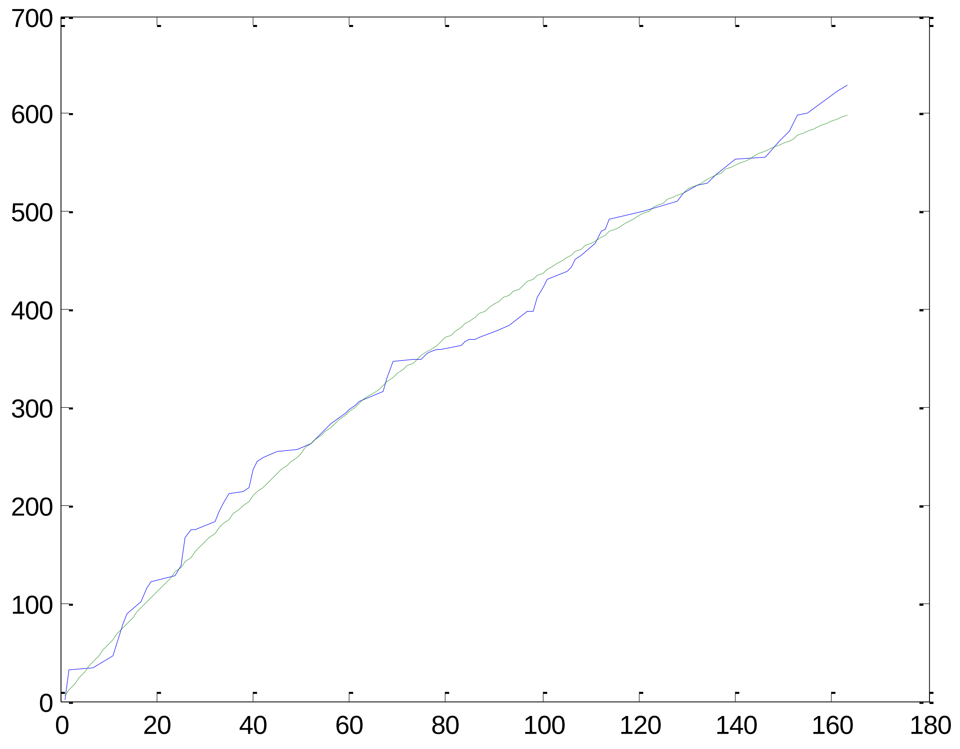
Time (days)

Figure 5.

From fig.4 analysis we see, that there were large time intervals without testing (20 and 15 days). Behavior of CB value after these intervals is very significant – we see reduction of CB increasing– see fig. 5. But standard models could not take into account these situations, because they analyze CB changing only depending of CE changing – see fig. 6. It is clear, that direct analysis CB from CE doesn't take into account silents on CE!

Using of proposed model allow us to take into account these silents (by means of calculation of parameter CEM) and to get more accurate results – see fig.7

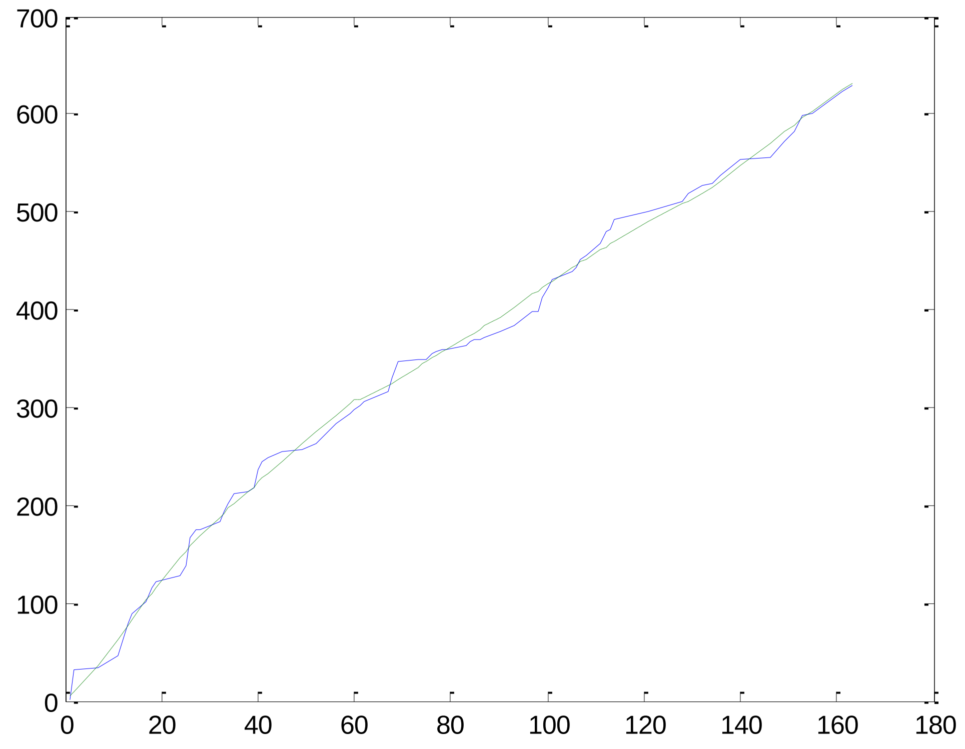
Cumulative Amount of Bugs



Cumulative Effort (person*day)

Figure 6 (------ results of measurements, - - - results of Ohba model calculation).

Cumulative Amount of Bugs



Cumulative Effort (person*day)

Figure 7 (------ results of measurements, - - - results of proposed model calculation).

Another drawback of classical Software Reliability model is following **- they don't take into account essential changing on projects during testing.**
See, for example following fig. 8 and 9. It is evident, that after value CE = 110 person*day (Day Number = 90 days) project essentially changed (new version and/or new tests were appeared) and so we don't see typical (concave) graph of classical Software Reliability model. Figure 8 shows the essentially trend of the cumulative number of bugs and stabilization of the product is not in sight. Fig. 9 illustrates changing of Cumulative Amount of All Bugs depending of Number of days. S-shape model isn't applicable for this case, it is rather double S-shaped model !!!

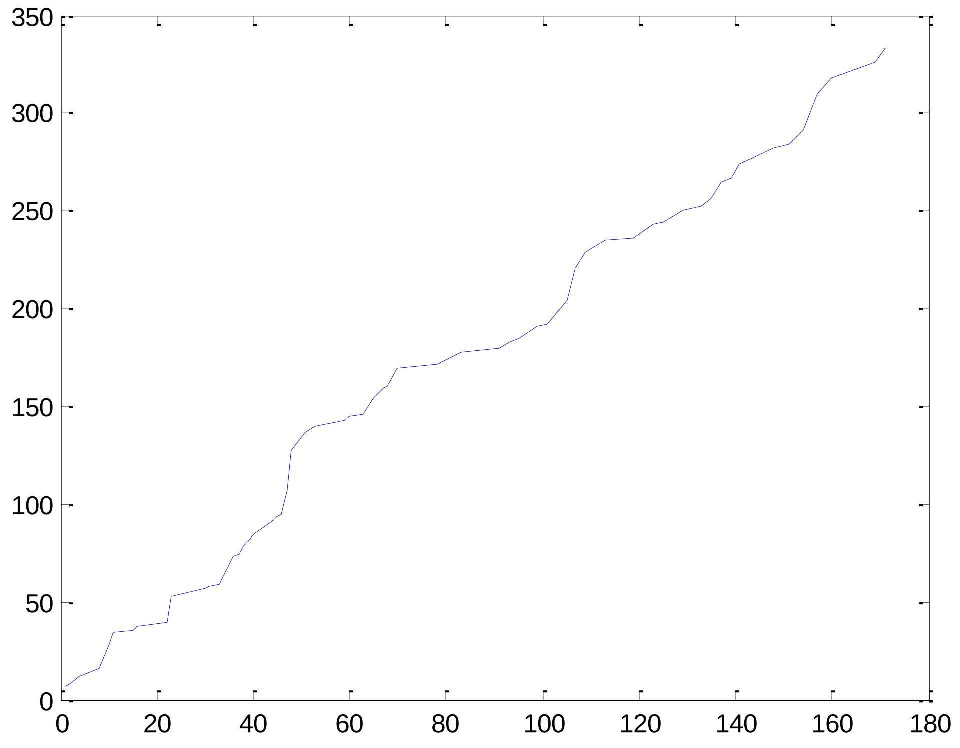
Suppose, that during testing following information may be obtained:

- n_ch[1], n_ch[2],… - number of days, corresponding for "essential" changing of software (e.g., days 88 and 116 from testing start)
- d_ch[1], d_ch[2],… - relative parts of software, that were changed (e.g. d_ch[1] = 1, d_ch[2] = 1.5).

Assume, that changing of Software follows for increasing of full Amount of Bugs (parameter N) for w_ch*d_ch[i]h %, where w_ch is unknown (control) parameter.

Modification on model take into account these parameters.

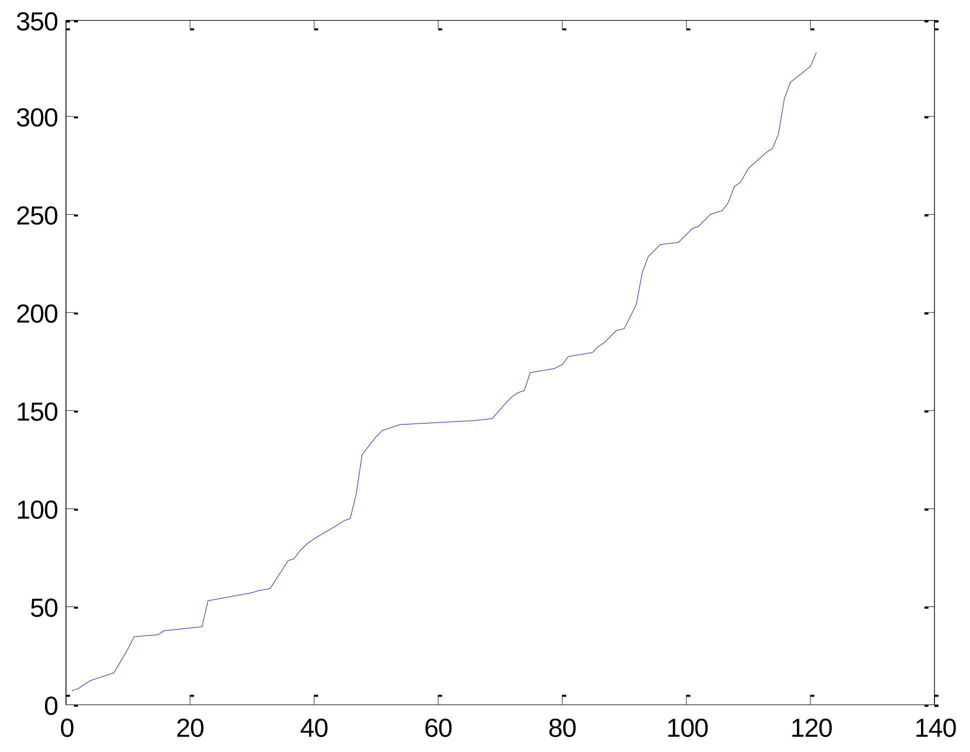
Cumulative Amount of All Bugs



Cumulative Effort (person*day)

Figure 8

Cumulative Amount of All Bugs



Day Number

Figure 9

After building of model type we have to define values of un-known (control) model parameters. In the case of "modified Ohba model" there are 8 parameters. Control parameters N, b, C, f, p and w_ch we define by means of Least Squares continuous Global Optimization, parameters TP and M are changed by means of simple discrete enumeration (usually TP = 7 or 14 days, M = 5, 15 or 30 days).

Global Optimization of non-linear function is a common task of a lot of practical problems. For example, concerning problem of Parameters Estimation, a Linear Regression model can support only a few cases (e.g., Duane Model). For other cases of non-linear regression using we have to search parameters by means of non-linear and non-convex, global optimization. Our task is to search value of **Z,** which provides

min G(**Z**) under constraints Low[i] <= z[i] <= High[i], i = 1…K, where:

- **Z =** {z[1],…,z[i],…z[K]} is a vector of parameters
- K is amount of parameters
- Low[i] is Low Boundary of Parameter i value (i = 1…K)
- High[i] is High Boundary of Parameter i value (i = 1…K)
- G is some Goal Function dependent of vector Z (in our case it is sum-of-difference-

squared between model-generated cumulative bug distribution curve to the actual cumulative bug distribution curve.

To solve this task, two different approaches can be used:

- To write and transform derivatives of Goal Function (i.e. Sum of Leased Squares for Non-Linear Regression method) for each single situation, to solve system of non-linear equations, corresponding these situations, to compare different obtained solutions, etc.
- To use "direct search methods", provided universal search of Global Minimum (without analytical definition of derivatives).

For first approach using we have to define complex analytical expressions for derivatives for each single situation. We propose to use second (**universal)** approach.

For second approach there are developed a lot of methods, based on gradient (or, if a goal function hasn't gradient – on pseudo-gradient) calculation and analysis. But for many real tasks the Goal Function isn't convex, it has many Local Minimums. In these cases such approaches require to know initial point of search, which has to be not far from optimal solution. But really we often don't know some information to define this initial point. So, it is impossible to use regular methods (gradients-based).

For Global Optimization Task we use one of the RANDOM SEARCH oriented methods – Cross-Entropy Optimization. It is relatively new approach, it isn't fully developed algorithm (as, for example, Genetic Algorithm, implemented as Toolbox on Matlab, or Simulated Annealing Algorithm), but it has provided good results for several analogous tasks.

3. Using of Proposed Model for Prediction

By means of proposed model we can compare different scenarios and predict best allocation of resources during testing on future. Single scenario we will describe by means of following parameters:

- Duration of Future Interval of Testing (days) – e.g., 15 days, 30 days, 60 days, etc.
- Amount of Cumulative Effort of Future Testing (person*days) – e.g., 30 person*days, 60 person*days, etc.
- Behavior of CE Rate depending of date number – e.g., constant, linear increasing from 0 to 2, linear decreasing from 4 to 0, etc.

Output parameter is Rest of Bugs after testing finishing (i.e. relative amount of bugs on the field).

Results of numerical calculations for some real projects are on the following table:

Scenario for Future			Output Parameters		
Duration of Interval	Amount of Cumulative Effort	Behavior of CE Rate	CE at the End	CEM at the End	Rest of the Bugs at the End (%)
30	30	constant = 1	193	230	10.5
30	30	0 -------> 2	193	223	11.5
30	30	2 -------→ 0	193	248	8.5
60	30	constant=0.5	193	211	13.0
60	30	0 -------> 1	193	208	13.5
60	30	1 -------→ 0	193	222	11.5
15	30	constant = 2	193	260	7.5
15	30	0 -------> 4	193	248	8.5
15	30	4 -------→ 0	193	276	6.0
30	60	constant = 2	223	329	3
30	60	0 -------> 4	223	325	3.5
30	60	4 -------→ 0	223	379	1.5
60	60	constant = 1	223	260	7.5
60	60	0 -------> 2	223	263	7
60	60	2 -------→ 0	223	293	5
15	60	constant = 4	223	423	1

Numerical results allow us to do some interesting conclusions:

- Although most significant factor is "Amount of Cumulative Effort of Future ", the parameter " Behavior of CE Rate" is also very significant – e.g., additional CE of 30 person*days for 15 days is more better, than additional CE of 60 person*days for 60 days! See, e.g., fig. 10 and 11.
- Scenario of reduction of CE rate always better, than scenario with CE rate

increasing (with same additional value of CE!).

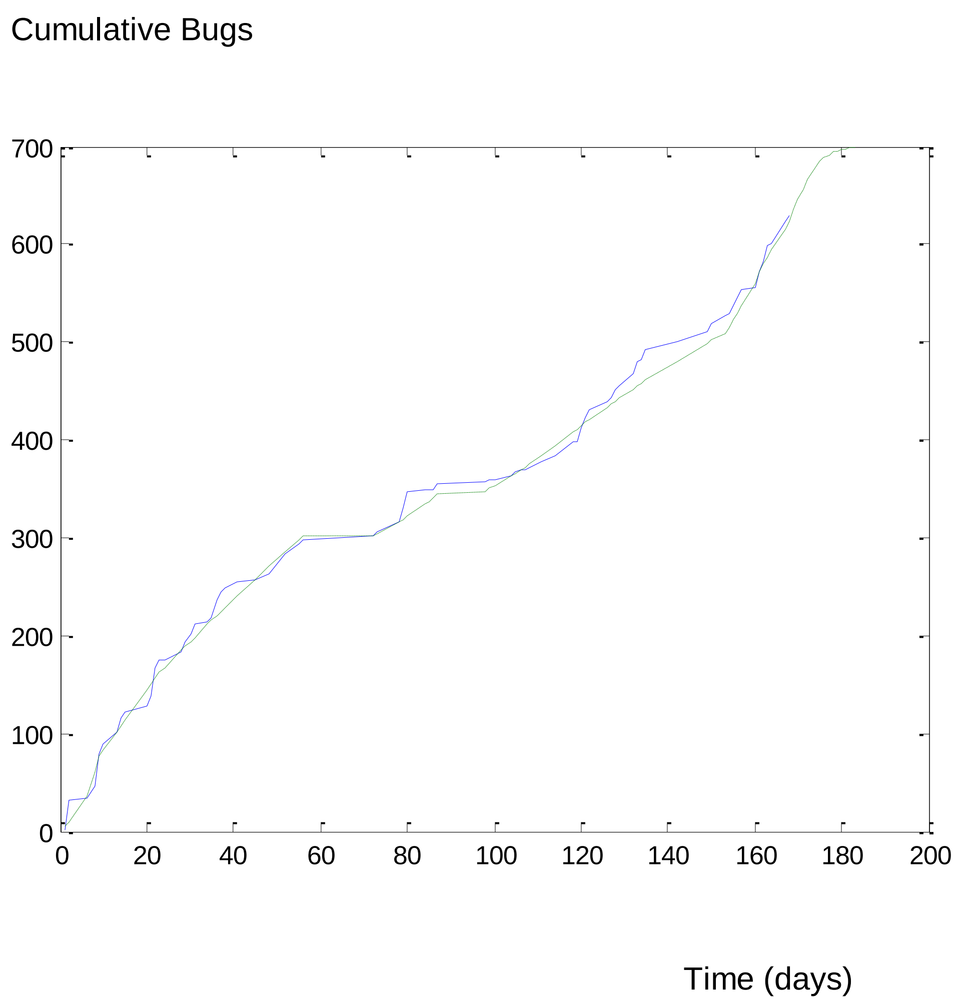
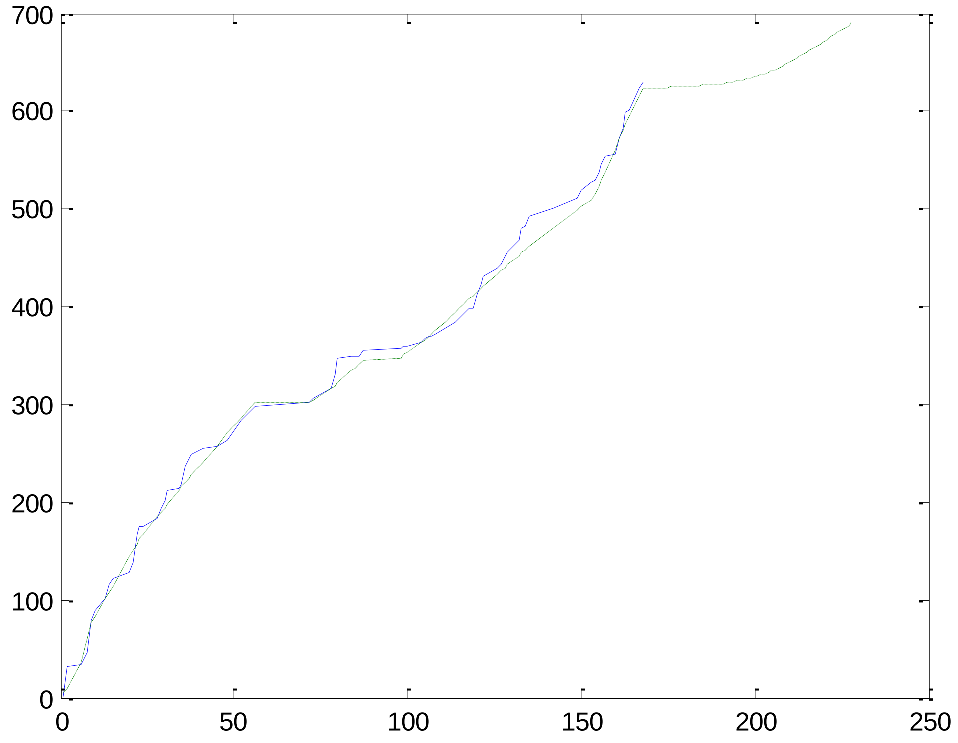


Figure 10. Scenario " Duration=15 days, CE Amount = 30 person*days, CE Rate = 4…0"

Cumulative Bugs



Time (days)

Figure 11. Scenario " Duration=60 days, CE Amount = 60 person*days, CE Rate = 0…2"