Software Measurement: Uncertainty and Causal Modelling

Norman Fenton[1], Paul Krause[2] and Martin Neil[1]
[1]Queen Mary, University of London and Agena Ltd.
Mile End Road
London E1 4NS

[2]Philips Research Laboratories
Crossoak Lane
Redhill
Surrey RH1 5HA

1st March 2001

Abstract

Software measurement has the potential to play an important role in risk management during product development. Metrics incorporated into predictive models can give advanced warning of potential risks. However, the common approach of using simple regression models, notably to predict software defects, can lead to inappropriate risk management decisions. These naïve models should be replaced with predictive models incorporating genuine cause-effect relationships. We show how these can be built using Bayesian networks; a powerful graphical modelling technique. We describe how a Bayesian network for software quality risk management is providing accurate predictions of software defects in a range of real projects. As well as their use for prediction, Bayesian networks can also be used for performing a range of "what if" scenarios to identify potential problems and possible improvement actions. This really is the dawn of an exciting new era for software measurement.

Introduction

Although "risk" tends to be associated most with safety critical systems, many now recognise that there will be uncontrolled risks to an organisation's reputation or profitability unless software development is carefully managed. We believe that software metrics are typically used in a way that militates against effective risk management. In this paper we will explain why this is so, and discuss what can be done to improve the situation.

Two specific roles of software measurement are quality control and effort estimation. Both of these fit under the general umbrella of risk management: what is the risk in terms of field call rates of releasing a product; and what is the risk of committing to a project given certain resources and certain commercial deadlines? Normally, these risks will have to be assessed and managed using predictive models. However, we argue that inappropriate risk management decisions will be made unless these models are based on sound causal relationships.

First, let us concentrate on software quality. Using the widely accepted definitions in [Fenton and Pfleeger, 1996], there are two different viewpoints of software quality. The first, the external product view, looks at the characteristics and sub-characteristics that make up the user's perception of quality in the final product – this is often called quality-in-use. Quality-in-use is determined by measuring external properties of the software, and hence can only be measured once the software product is complete. For instance quality here might be defined as freedom from defects or the probability of executing the product, failure free, for a defined period.

The second viewpoint, the internal product view, involves criteria that can be used to control the quality of the software as it is being produced and can form early predictors of external product quality. Good development processes and well-qualified staff working on a defined specification are just some of the pre-requisites for producing a defect free product. If we can ensure that the process conditions are right, and can check intermediate products to ensure this is so, then we can perhaps produce high quality products in a repeatable fashion.

Unfortunately the relationship between the quality of the development processes applied and the resulting quality of the end products is not deterministic. Software development is a profoundly intellectual and creative design activity with vast scope for error and for differences in interpretation and understanding of requirements. The application of even seemingly straightforward rules and procedures can result in highly variable practices by individual software developers. Under these circumstances the relationships between internal and external quality are uncertain.

Typically informal assessments of critical factors will be used during software development to assess whether the end product is likely to meet requirements:

- Complexity measures: A complex product may indicate problems in the understanding of the actual problem being solved. It may also show that the product is too complex to be easily understood, de-bugged and maintained.
- Process maturity: Development processes that are chaotic and rely on the heroic efforts of individuals can be said to lack maturity and will be less likely to produce quality products, repeatedly.
- Test results: Testing products against the original requirements can give some indication of whether they are defective or not. However the results of the testing are likely only to be as trustworthy as the quality of the testing done.

The above types of evidence are often collected in a piece-meal fashion and used to inform the project or quality manager about the quality of the final product. However, there is often no formal attempt in practice to combine such evidence together into a single quality model. Instead, there is a focus on using “naïve” regression models as predictors of quality in use.

The problem with regression models

A holy grail of risk management could be the identification of one simple measurement that provides an advanced warning of there being a high likelihood that a potential hazard will be realised. In some cases, an established causal relationship is available. For example, current scientific and epidemiological evidence has established the link from smoking to lung cancer beyond all reasonable doubt. Thus a doctor can predict with some confidence that a smoker runs a relatively high risk of developing lung cancer compared to a non-smoker. Equally, a smoker can manage this risk by giving up smoking. Unfortunately, in software engineering the causal relationships are rarely so straightforward. We will illustrate this with one simple example. More detailed analyses of naïve regression models for software engineering can be found in Fenton and Neil [2000], and Fenton and Ohlsson [2000].

Suppose we have a product that has been developed using a set of software modules. A certain number of defects will have been found in each of the software modules during testing. Perhaps we might assume that those modules that have the highest number of defects during testing would have the highest risk of causing a failure once the product was in operation? That is, we might expect to see a relationship similar to that shown in figure 1.

<Figure 1 about here>

What actually happens? It is hard to be categorical. However, two published studies indicate quite the opposite effect – those modules that were most problematic pre-release had the least number of faults associated with them post-release. Indeed, many of the modules with a high number of defects pre-release showed zero defects post-release. This effect was first demonstrated by Adams [1984], and replicated by Fenton and Ohlsson [2000]. Figure 2 is an example of the sort of results they both obtained.

<Figure 2 about here>

So, how can this be? The simple answer is that faults found pre-release gives absolutely no indication of the level of residual faults *unless* the prediction is moderated by some measure of test effectiveness. In both of the studies referenced, those modules with the highest number of defects pre-release had had all their defects "tested out". In contrast, many of the modules that had few defects recorded against them pre-release clearly turned out to have been poorly tested – they were significant sources of problems in the final implemented system.

Starting to think about causality

An extreme example of the difficulties that can arise through the use of naïve regression models is what is known as Simpson's paradox. This was first recognised in 1899 by Karl Pearson. He noticed that a statistical relationship between two parameters could be reversed by the inclusion of an additional factor in the analysis. For example, suppose we have data that indicates that students who smoke obtain higher grades than students who don't smoke. Does this really express a causal relationship? If a student is at risk of failing an exam, should we encourage them to smoke?

<Table 1 about here>

The same set of data can support quite a different conclusion if age is taken into consideration, see Table 1. In general, older students tend to smoke more. In addition, the more mature students tend to score higher grades. Thus, age "causes" smoking, and age "causes" higher grades. However, *for any fixed age group*, smokers may obtain lower grades than non-smokers. Now if we have a student who is at risk of failing an exam, this model would suggest the advice that they take a maturer attitude to their studying.

Eliciting the "correct" causal model purely from statistical data is an unsolved problem in general. However, we can often use science and reason to identify the causal influences in a problem of interest. We believe that this is an important first step that has so far almost always been overlooked when producing predictive models in software engineering. The example from the preceding section is a case in point. We cannot make judgements of the quality of software from defect data alone. We must also take into account, at least, the effectiveness with which the software has been tested.

<Figure 3 about here>

Figure 3 provides a slightly more comprehensive model. "Defects Present", is the attribute we are interested in. This will have a causal influence on the number of "Defects Detected" during testing. "Test Effectiveness" will also have a causal influence on the number of defects detected and fixed. As we will see later, this will turn out to be a fragment of a much larger model, with the node representing defects present being a synthesis of a number of factors including, for example, review effectiveness, developer's skill level, quality of input specifications and resource availability.

For this discussion, we will assign two states to each node; "low" and "high". We can now think of the qualitative behaviour that we would expect from such a model. Straightforward reasoning from cause to effect is possible. So, if Test Effectiveness is "low" then Defects Detected would also be "low". But Defects Detected would also be low if Defects Present was "low".

One of the beauties of the causal models that we are building is that they enable us to reason in both a "forward" and a "reverse" direction. That is, we can identify the possible causes given the observation of some effect. In this case, if the number of defects detected is observed to be “low” the model will tell us that low test effectiveness and a low number of defects present are both possible explanations (perhaps with an indication as to which one is the most likely explanation). The concept of “explaining away” can also be modelled. For example, if we also have independent evidence that there are indeed a low number of defects present, then this will provide sufficient explanation of the observed value for defects detected and the likelihood that test effectiveness was low will be reduced.

Making the models work

So far we have only provided a static description of a causal model, and then a qualitative description of its dynamic behaviour. We clearly need to augment the graphical model with a calculus that enables us to update the model as new evidence is obtained. We do this by assigning probability tables to the nodes in the graphical model, and then using Bayes’ theorem to revise the probabilities as we obtain new information about a specific problem.

The notion of conditional probability is fundamental to this approach. We will write the probability of some event E, given some condition C, as: p(E | C). Typically, these probabilities are easiest to elicit if C represents some cause and E represents some effect of that cause. For example, the cause might be measles, and the effect red spots. The “likelihood” p(red spots | measles) could be elicited by counting all those patients with measles who exhibit red spots. It might seem a little paradoxical to assign a probability, a measure of *uncertainty*, to a causal relationship. After all, causality is usually seen as deterministic. However, these causal rules are being expressed at a macroscopic level. The microscopic details of exceptional events, interruptions to the flow of causality, are in effect summarised out by assigning this numerical probability that gives a measure of how likely this inference is to be valid.

For simplicity, we will look at only one branch of the model in Figure 3. Let variable DD represent “Defects Detected” and variable TE represent “Test Effectiveness”. Suppose that we had elicited the likelihood p(DD = “high” | TE = “high”) = 0.8. What we are now interested in is the “posterior probability” p(TE = “high” | DD = “high”)? That is, if we observe that a large number of defects have been detected, what is the probability that the software item has been effectively tested? To calculate this, we need two additional pieces of information. Firstly, the “prior” probability p(TE = “high”) – a measure of what is typically the case. We will take p(TE = “high”) = 0.2 in this example. Secondly, we need p(DD = “high”), again a measure of what is typically the case. Just for illustration, in this case we might express a degree of ignorance and say, “well it is just as likely for the number of defects detected to be high as low”. That is, p(DD = “high”) = 0.5.

Now, suppose we observe that for a specific product, the number of defects detected is “high”. What is the probability in this case, that the product was well tested? This can now be calculated using Bayes’ rule. This is stated in general terms as:

$$p(E \mid C) = \frac{p(C \mid E)\, p(E)}{p(C)}$$

In our example,

$$p(TE = \text{"}high\text{"} \mid DD = \text{"}high\text{"}) = \frac{p(DD=\text{"}high\text{"} \mid TE=\text{"}high\text{"})\, p(TE=\text{"}high\text{"})}{p(DD=\text{"}high\text{"})}$$

$$= \frac{0.8 \times 0.2}{0.5} = 0.32$$

We see that the definite observation of a high number of defects has increased the probability that the product was effectively tested, although not by a dramatic amount.

This approach can be generalised to update evidence over sets of variables, not just the two used in this simple example. In addition, variables with more than two discrete states, or even continuous variables, are permissible. However, two major difficulties needed to be overcome before these *Bayesian Networks*, the combination of causal graphical models with node probability tables, could become a useable tool. Firstly, Bayesian updating as outlined above is extremely computationally intensive in the general case. Secondly, eliciting all the node probability tables is an intractable knowledge elicitation problem for anything but the most trivial of problems.

Fortunately, a number of breakthroughs have been made in the last twenty years that address both of these problems. Firstly, fast update algorithms have been developed that exploit topological properties of the graphical models to reduce the computational complexity [Lauritzen & Spiegelhalter, 1988; Pearl, 1988]. These are available in a number of commercial inference engines. Secondly, techniques for modularising large Bayesian Networks together with graphical tools for editing the probability distributions in the node probability tables, enable the knowledge elicitation to be performed using limited resources [Neil, Fenton and Nielson, 2000].

A Bayesian Network for defect prediction

We have built a module level defect prediction model in order to explore these ideas further. The resulting Bayesian Network takes into account a range of product and process factors from throughout the lifecycle of a software model. It has been evaluated against real project data, and a detailed description of the model together with the results of the evaluation can be found in [Fenton, Krause and Neil, *submitted for publication*]. What we propose to do here is to demonstrate the tool in action to illustrate how it handles the relationship between defects detected during test, and residual defects delivered.

The tool is called AID, for *Assess, Improve, Decide*. The “Intrinsic Complexity” of the problem being solved is included amongst the attributes represented in the Bayesian Network. This has five states, ranging from “very simple” to “very complex”. We will first take a look at how the number of defects detected during unit test varies as the intrinsic complexity is altered between these two extremes. Figure 4 shows the prior probability distribution for defects detected and fixed. The horizontal axis is labelled with interval values for the number of defects detected and fixed. The vertical axis indicates the probability that the number of defects will fall within a certain interval. This distribution has a median of 90 (the distributions in this example are skewed, so we will take the median value as the most appropriate summary statistic). Note that as we have not entered any information about a specific module, the distribution indicates a very wide range of possible values.

<Figure 4 about here>

Now, we enter the information that the intrinsic complexity of the software module is “very complex”. Figure 5a shows the new distribution for the number of defects detected during unit test. Note that the distribution has shifted to the left, and the median value has *dropped* to 30. At first sight this seems counter intuitive. Especially when this is compared with Figure 5b, which shows the prediction for a “very simple” module. In the latter case, the distribution shifts to the right, with the median *increasing* to 125.

<Figure 5 about here>

The explanation is that the more complex modules are harder to test than the simpler modules. With their greater ability to “hide” faults, fewer faults will be detected unless there is a compensating increase in the effectiveness with which the module is tested. No such compensation has been applied in this case and the low prediction for defects detected and fixed for the “very complex” case indicates that typically such modules are relatively poorly tested.

This is borne out when we look at the respective figures for residual defects delivered (Figure 6). Now we see a reversal. The prediction for the “very complex” module indicates that it will contain more residual defects than the “very simple” module (a median of 70, compared to a median of 30). So our model naturally produces the qualitative behaviour of the real world data from our earlier experiment. That is, the better-tested modules yield more defects during unit test and deliver fewer defects. For the more poorly tested modules, the converse is the case.

<Figure 6 about here>

Conclusion

Although software measurements play an important role in process control and risk management, their full potential has yet to be realised. Part of the reason for this is the reliance that is placed on the use of simple regression models that fail to take into account the major causal influences on the quality goals of a project.

We have shown that there really is no longer any excuse against building predictive models that can support effective risk management decisions. Bayesian networks can be used to construct models that encapsulate causal influences on a development project. As well as their use for prediction, they can also be used for performing “what if” scenarios in order to identify potential hazards if some aspect of a process underperforms, or to identify possible improvement actions if a current prediction fails to meet a target. This really is the dawn of an exciting new era for software measurement.

References

E. Adams, “Optimizing preventive service of software products”, *IBM Research Journal*, **28**(1), 2-14, 1984.

N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, (2nd Edition), PWS Publishing Company, 1997.

N.E. Fenton and M. Neil “A Critique of Software Defect Prediction Research”, *IEEE Trans. Software Eng*., **25**, No.5, 1999.

N.E. Fenton and N. Ohlsson “Quantitative analysis of faults and failures in a complex software system”, *IEEE Trans. Software Eng.*, **26**, 797-814, 2000.

N.E. Fenton, P.J. Krause and M. Neil, “A Probabilistic Model for Software Defect Prediction”, *submitted for publication*, 2001.

S.L. Lauritzen and D.J. Spiegelhalter, “Local computations with probabilities on graphical structures and their application to expert systems (with discussion)” *J. Roy. Stat. Soc.* **Ser B 50**, pp. 157-224, 1988.

M. Neil, N. Fenton and L. Nielson, “Building large-scale Bayesian Networks”, *Knowledge Engineering Review*, *to appear* 2000.

J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* Morgan Kauffman, 1988 (Revised in 1997).

Biographical sketches and Full Contact Details

Norman Fenton

Department of Computer Science,
Queen Mary, University of London,
Mile End Road, London E1 4NS
United Kingdom
Tel: +44 (0) 20 7882 7860
Fax: +44 (0) 1223 263899
E-mail: norman@dcs.qmw.ac.uk

and

Agena Ltd.
11 Main Street, Caldecote,
Cambridge CB3 7NU

Norman Fenton is Professor of Computing at Queen Mary (University of London) and is also Managing Director of Agena, a company that specialises in risk management for critical systems. Between 1989 and March 2000 he was Professor of Computing Science at the Centre for Software Reliability, City University. Norman is a Chartered Engineer (member of the IEE) and a Chartered Mathematician (Fellow of the IMA). He has been project manager and principal researcher in many major collaborative projects in the areas of: software metrics; formal methods; empirical software engineering; software standards, and safety critical systems. His recent research projects, however, have focused on the use of Bayesian Belief nets (BBNs) and Multi-Criteria Decision Aid for risk assessment. Also, Agena has been building BBN-based decision support systems for a range of major clients. Norman's books and publications on software metrics and formal methods are widely known in the software engineering community. Norman is a member of the Editorial Board of both the Software Quality Journal and the Journal of Empirical Software Engineering, and is on the Advisory Board of Q-Labs.

Paul Krause

Philips Research Laboratories,
Crossoak Lane, Redhill
Surrey RH1 5HA
Tel: +44 (0) 1293 815298
Fax: +44 (0) 1293 815500
E-mail: paul.krause@philips.com

Paul Krause has a varied background in software engineering. His areas of expertise include requirements management, formal specification, theoretical aspects of artificial intelligence, and software process and product quality assessment and improvement. He now works in the Specification and Testing Cluster of the Software Engineering and Applications Group at Philips Research Laboratories, Redhill, UK. Current projects address automatic test case execution for embedded systems, requirements management using the UML methodology, and software reliability prediction. Paul is also on the British Computer Society's

ISEB Software Testing Board. At the beginning of 2001, Paul was appointed to work part of his time as Research Professor in Software Engineering at the Department of Computer Science, Surrey University. Paul is a Chartered Mathematician (Fellow of the IMA).

Martin Neil

Department of Computer Science,
Queen Mary, University of London,
Mile End Road, London E1 4NS
United Kingdom
Tel: +44 (0) 20 7882 5221
Fax: +44 (0) 1223 263899
E-mail: martin@dcs.qmw.ac.uk

and

Agena Ltd.
11 Main Street, Caldecote,
Cambridge CB3 7NU

Martin Neil is a Lecturer in the Computer Science Department of Queen Mary and Westfield College, University of London. He holds a degree in 'Mathematics for Business Analysis' from Glasgow Caledonian University and a PhD in ‘Statistical Analysis of Software Metrics’ jointly from South Bank University and Strathclyde University. From 1996 - 1999 he spent four years at the Centre for Software Reliability, City University (London) and from 1992 - 1995 worked for Lloyd's Register as a consultant. His interests cover applications and theory in software metrics, Bayesian probability, reliability and the software process. Martin is a Charted Engineer (IEE), a member of the CSR Council, the IEEE Computer Society and the ACM. Martin is also a director of Agena, a consulting company specialising in decision support and risk assessment of safety and business critical systems: www.agena.co.uk. (Contact: martin@agena.co.uk)

Figures

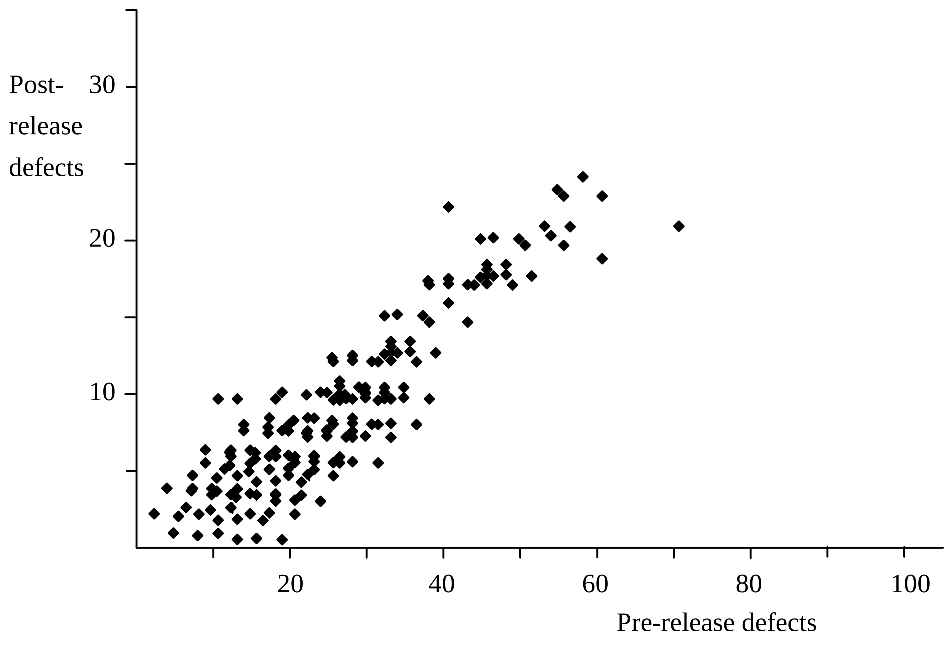


Figure 1: A hypothetical plot of pre-release against post-release defects for a range of modules. Each dot represents a module.

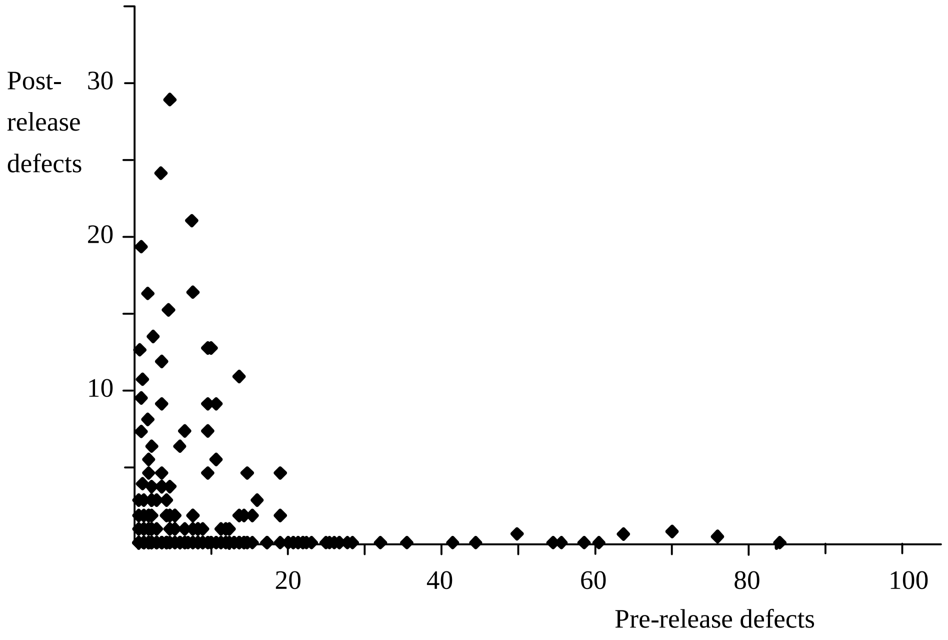


Figure 2: Actual plot of pre-release against post-release defects for a range of modules.

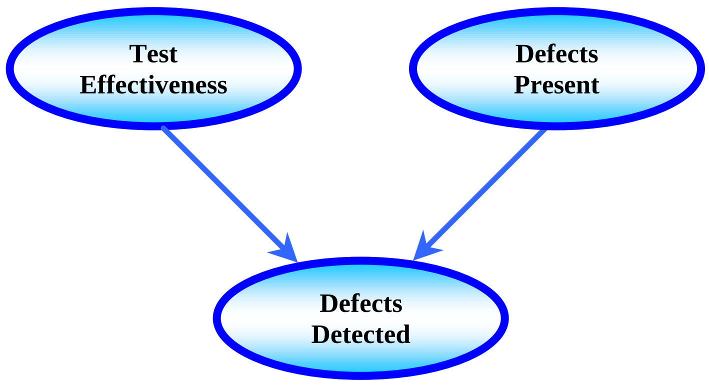


Figure 3: A simple graphical model that provides greater explanatory power than a naïve regression model.

	Non-smoker	**Smoker**
High Marks	1095	5050
Low Marks	9005	5950

Table 1a: With the above grouping of data, approximately 50% of smokers get high marks, whilst only 10% of non-smokers do.

	Age 1		**Age 2**	
	Non-smoker	**Smoker**	**Non-smoker**	**Smoker**
High Marks	1000	50	95	5000
Low Marks	9000	950	5	5000

Table 1b: However, a very different picture emerges if we take into account the age structure. Here, we take a simple classification with Age 1 < Age 2. Now we see that for age group 1, approximately 5% of smokers get high marks but 10% of non-smokers do. Similarly, for age group 2, 50% of smokers get high marks but a significantly larger 95% of non-smokers get high marks. So for both groups, non-smokers significantly outperform smokers. The data in Table 1a was dominated by the massive increase in smoking in age group 2. Although this is a simple manufactured example, the same effect can easily be observed with real data.

Table 1: Two different ways of clustering the same data

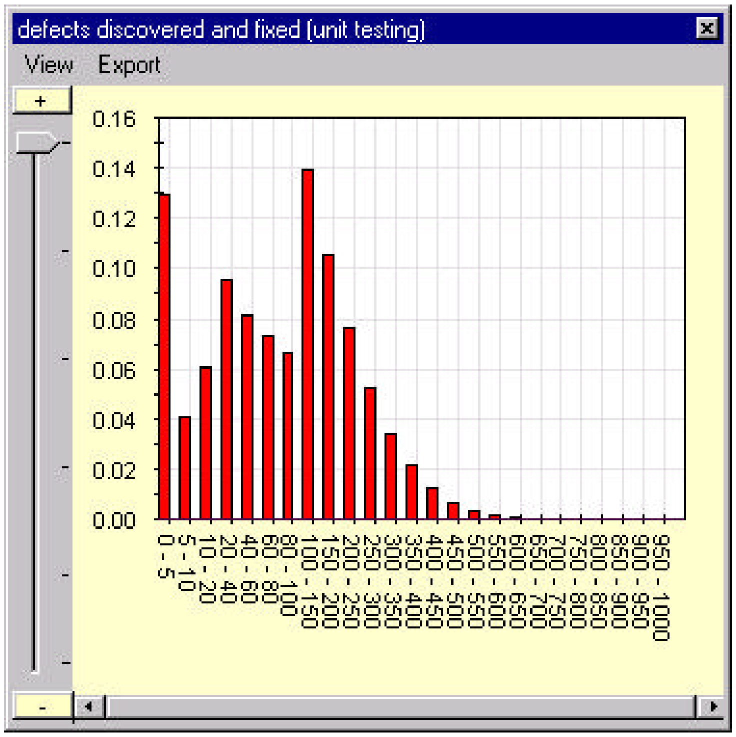
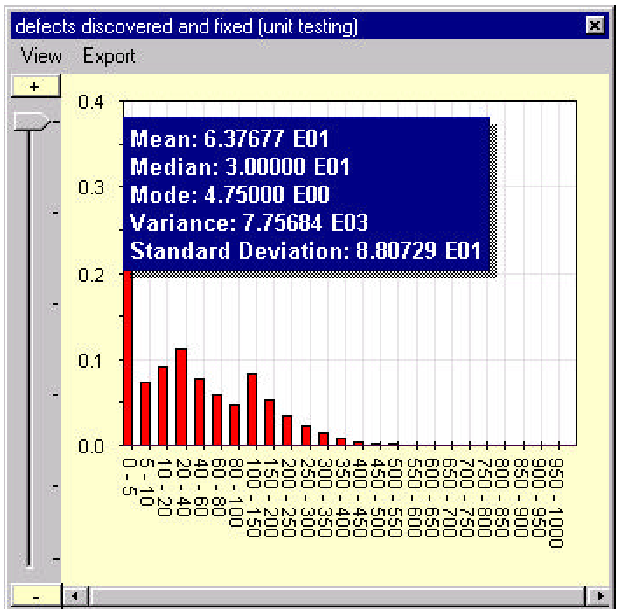
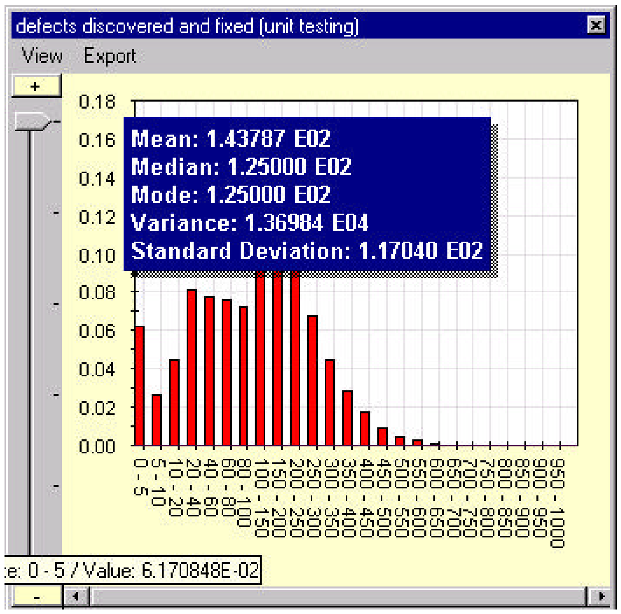


Figure 4: Prior distribution for defects detected and fixed during unit test.



Mean: 6.37677 E01
Median: 3.00000 E01
Mode: 4.75000 E00
Variance: 7.75684 E03
Standard Deviation: 8.80729 E01

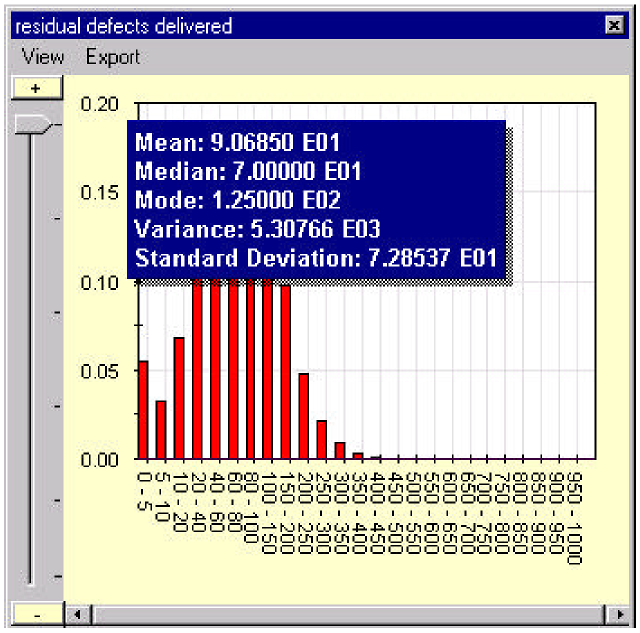


Mean: 1.43787 E02
Median: 1.25000 E02
Mode: 1.25000 E02
Variance: 1.36984 E04
Standard Deviation: 1.17040 E02

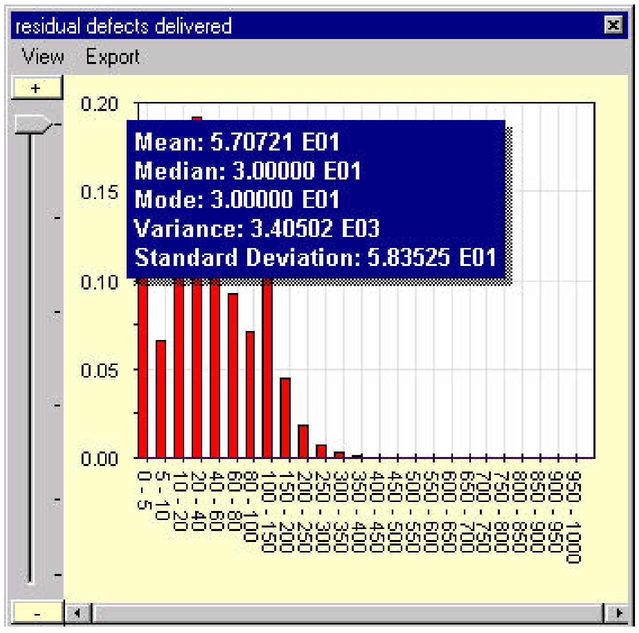
:e: 0 - 5 / Value: 6.170848E-02

(a) (b)

Figure 5: Distribution for defects detected at unit test for a "very complex" module (a), compared with that for a "very simple" module (b).



Mean: 9.06850 E01
Median: 7.00000 E01
Mode: 1.25000 E02
Variance: 5.30766 E03
Standard Deviation: 7.28537 E01



Mean: 5.70721 E01
Median: 3.00000 E01
Mode: 3.00000 E01
Variance: 3.40502 E03
Standard Deviation: 5.83525 E01

(a) (b)

Figure 6: Distribution for residual defects delivered for a “very complex” module (a), compared with that for a “very simple” module (b).