UCRL-ID-138577

Conceptual Software Reliability Prediction Models for Nuclear Power Plant Safety Systems

April 3, 2000

Prepared by
Gary Johnson, LLNL and University of California at Berkeley
J. Dennis Lawrence
Xingbo Yu, University of California at Berkeley

Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

Prepared for
U.S. Nuclear Regulatory Commission

This work was supported by the United States Nuclear Regulatory commission under a Memorandum of Understanding with the United States Department of Energy, and performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Conceptual Software Reliability Prediction Models for Nuclear Power Plant Safety Systems

Prepared by
Gary Johnson, LLNL and University of California at Berkeley
J. Dennis Lawrence
Xingbo Yu, University of California at Berkeley

Prepared for
U.S. Nuclear Regulatory Commission

FESSP
Fission Energy and Systems Safety Program
Lawrence Livermore National Laboratory

CONTENTS

FIGURES

TABLES

EXECUTIVE SUMMARY

LLNL evaluated a number of reliability prediction models for possible use by the NRC in evaluating nuclear power plant safety systems. Two promising models emerged from a series of candidate measures previously identified as the most plausible indicators of software reliability. One model uses Bayesian Belief Networks (BBN) to model the influence of process and product qualities upon reliability and upon the software measures. The second model is a modification of an existing model developed by Rome Laboratory. The parameters of the Rome Laboratory model were estimated based upon the previously identified software engineering measures. Both models depend upon knowing what information the selected software measures provide about software reliability. These relationships are difficult to ascertain without access to development information. The relationships could be developed if sufficient project data were available. Both models will require extensive validation before the numerical predictions they generate could be considered credible.

The BBN models the factors that influence software reliability and the influence of these factors upon the software measures identified by a previous task. As evidence about the software and the software development process is gathered in the form of measures, it is entered into the model to update the reliability prediction. The BBN model considers requirements, architectural design, detailed design, implementation, and validation testing activity groups. It produces a discretized prediction of the reliability distribution for the software under consideration. The BBN model appears to provide a method that can be used to combine software measures with NRC inspection results to produce a composite measurement of software quality. This may offer a way to improve the quality and repeatability of NRC software audits. The BBN model presented in this report essentially models the thought process of an auditor applying the review guidance of the Standard Review Plan. It offers a method for combining judgements from qualitative assessments (e.g., review according to BTP-14) with quantitative measures for possible improvement of the NRC software review process. This path appears to be a viable route to developing assessment tools that can be useful to the NRC staff and that could be applied in many other user domains.

The Rome Laboratory model uses checklists to develop quality factors used to adjust a base failure rate established for the given type of software. The combinatorial model developed in this report is equivalent to the Rome Laboratory model except that the quality factors are estimated based upon the previously identified candidate software measures. The combinatorial model does not explicitly consider different development activity groups. It produces a point estimate of software reliability.

LLNL developed the models to the point where they are ready to be tested. Although neither should be considered a credible method for software reliability prediction at this time, they both offer possible approaches that could be further developed. The models could be tested, modified as necessary based on the tests, and validated.

CONCEPTUAL SOFTWARE RELIABILITY PREDICTION MODELS FOR NUCLEAR POWER PLANT SAFETY SYSTEMS

1. INTRODUCTION

1.1 Project Objective

The objective of this project is to develop a method to predict the potential reliability of software to be used in a digital system instrumentation and control system. The reliability prediction is to make use of existing measures of software reliability such as those described in IEEE Std 982 and 982.2[1][2]. This prediction must be of sufficient accuracy to provide a value for uncertainty that could be used in a nuclear power plant probabilistic risk assessment (PRA). For the purposes of the project, *reliability* was defined to be the probability that the digital system will successfully perform its intended safety function (for the distribution of conditions under which it is expected to respond) upon demand with no unintended functions that might affect system safety.

The ultimate objective is to use the identified measures to develop a method for predicting the potential quantitative reliability of a digital system. The reliability prediction models proposed in this report are conceptual in nature. That is, possible prediction techniques are proposed and trial models are built, but in order to become a useful tool for predicting reliability, the models must be tested, modified according to the results, and validated.

Using methods outlined by this project, models could be constructed to develop reliability estimates for elements of software systems. This would require careful review and refinement of the models, development of model parameters from actual experience data or expert elicitation, and careful validation. By combining these reliability estimates (generated from the validated models for the constituent parts) in structural software models, the reliability of the software system could then be predicted.

Modeling digital system reliability will also require that methods be developed for combining reliability estimates for hardware and software. System structural models must also be developed in order to predict system reliability based upon the reliability of the individual hardware/software components. Existing modeling techniques — such as fault tree analyses or reliability block diagrams — can probably be adapted to bridge the gaps between the reliability of the hardware components, the individual software elements, and the overall digital system.

This project builds upon previous work to survey and rank potential measurement methods which could be used to measure software product reliability[3]. This survey and ranking identified candidate measures for use in predicting the reliability of digital computer-based control and protection systems for nuclear power plants. Additionally, information gleaned from the study can be used to supplement existing review methods during an assessment of software-based digital systems.

1.2 Project Purpose

The ultimate purpose of the method developed to predict potential reliability of digital safety systems is to provide:

- a reliability prediction along with the associated uncertainty which can be used in a PRA. These values would be used as input to the PRA model to approximate the probability of success or failure of the digital system to perform its intended function under accident conditions

  and

- information that can be used to supplement existing review methods in an assessment of software-based digital systems as part of a staff review of (1) applications for nuclear reactor licenses or permits, (2) amendments to existing licenses, and (3) NRR user needs.

These purposes require tentatively postulated order-of-magnitude provisional predictions of digital system reliability. (That is, a conclusion that the eventual digital system failure rate is more likely to be approximately 10^{-3} than either 10^{-2} or 10^{-4} is considered satisfactory.)

These purposes also require a reliability prediction before the digital system is available for actual testing, because (1) design changes are much simpler and less expensive to make early in a system's life cycle; (2) it is impossible to create a system with a given reliability level if the reliability level cannot be assessed until the system is already created; and (3) NRC cannot evaluate an applicant's proposed design if a reliability prediction must wait until after the complete system is built and tested to be useful. Consequently, development of a model for estimating potential reliability, in addition to statistical testing and operational experience, is desirable. This model can be used early in the digital system development lifecycle, and may possibly be combined to provide an estimate of digital system reliability that is better than any single method.

1.3 Background

Regulatory Guide 1.152, Revision 1[4], which endorsed IEEE Standard 7-4.3.2, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations,"[5] stated that the staff does not endorse Section 5.15, "Reliability," as a sole means of meeting the Commission's regulations for reliability of digital equipment used in safety systems. Section 5.15 of the standard states, "when qualitative or quantitative reliability goals are required, the proof of meeting the goals shall include software used with hardware." The NRC staff did not endorse that section because there is no general agreement that a measurement methodology currently exists that provides a credible method to predict software reliability. Measurement is useful, but it cannot be the sole criterion to predict reliability.

During the last several years, both the NRC and the nuclear industry have recognized that PRA analysis has evolved to the point where it can be used as a tool for assisting regulatory decision making. In 1995, the NRC adopted a policy regarding expanded NRC use of PRA. Following publication of the Commission policy, the Commission directed the NRC staff to develop a regulatory framework that incorporates risk insights. Recently, NRC staff has developed risk-informed regulatory guides to meet this directive. PRAs require a value for failure-rate-per-demand to perform its intended function. Digital systems in nuclear power plant safety applications are expected to have extremely low failure rates ($< 10^{4}$ failures per demand), comparable to those for the existing hardwired systems that the digital systems will replace. However, no credible methodology currently exists to assess failure rates of this order for digital systems, so digital systems cannot be considered in any meaningful way when developing these risk insights. The development of objective predictions for digital system software reliability which could be used in place of subjective estimates in PRAs, where feasible, will help the NRC make better risk-informed decisions.

1.4 Statement of the Problem

Calculating the reliability of a hardware device is a well-understood problem. Robust methods have been developed to calculate hardware reliability, and used for many years when constructing PRA models. However, the techniques cannot easily be carried over to digital systems, which involve complex interactions between software and hardware.

Hardware reliability calculation starts with the assumption that the device design is fundamentally correct. Failures are assumed to be caused by random variability in the physical stresses seen by the device and by variability in the devices' ability to withstand these stresses. Consequently, the failure rate of a hardware device can be calculated from the failures observed in similar devices subject to similar stresses.

Software is a logical, not a physical, component of a system. Therefore, its success or failure is not affected by physical stress. Failures stem from fundamental errors in the design that cause the system to fail under certain combinations of system states and input trajectories. Reliability prediction for digital systems must account for the effects of hardware and software failures on each other.

In a 1993 study of the problem to quantify software reliability, Ricky Butler and George Finelli stated, "the quantification of life-critical software reliability is infeasible using statistical methods whether applied to standard software or fault-tolerant software." The paper showed that the classical methods of estimating reliability lead to exorbitant amounts of testing when applied to life-critical software. In addition, the paper examined reliability growth models and showed that they are incapable of overcoming the need for excessive amounts of testing. The key assumption of software fault tolerance—that separately programmed versions fail independently—was shown to be problematic.[6]

Because of the limitations discussed in the previous paragraphs, even statistical testing of the final system is incapable of producing realistic reliability predictions for highly dependable systems. Using such data may lead PRA studies to overestimate the contribution of digital system failures to total risk. Consequently, even where statistical testing is performed, a different approach to predicting digital system reliability is needed.

1.5 Approach

It is clear that, at the current state-of-the-practice, no single metric exists that can be utilized to predict software reliability prior to the availability of the final system. The primary method available when the final system is available (statistical testing) is not feasible in many cases where the demonstration of very high levels of reliability (e.g., greater than 0.999) is desirable. Therefore, the NRC requested that a different approach be investigated by this research. It was postulated that some selected "set" of measures might be combined by some algorithm to provide a better early prediction than any single qualitative measure currently used by industry. The approach outlined below was structured to test this theory.

The project consisted of the following five individual tasks structured in a sequence to develop and test an algorithm that would construct a prediction of digital system reliability and a value for the associated uncertainty for the prediction.

TASK 1— Investigate current software product reliability measurement methods.

This task was intended to perform a complete survey of all potential measures that might be of use in estimating software reliability. After completion of the survey, these measures were to be ranked to arrive at the most reasonable set for which an algorithm can be developed.

TASK 2— Develop conceptual software reliability measurement methodology based upon selective use of current software product reliability measurement methods.

This task was to develop one or more algorithms that can be used to predict software reliability to within an order of magnitude.

TASK 3— Test the conceptual software reliability measurement methodology.

This task was to perform the measurements required for the algorithms developed in the previous task and compare the results to field data.

TASK 4— Develop the digital system algorithm, incorporating conceptual software reliability measurement methodology.

This task was intended to integrate the resultant algorithms developed in Task 2 with hardware algorithms to form an integrated digital system measurement technique.

TASK 5— Test the digital system measurement algorithm.

This task was to perform the measurements required by the algorithm developed in the previous task and compare the results to field data.

1.6 Task 2 Summary

This report documents the results of the Task 2 development of models to predict software product reliability using the candidate software reliability measures identified in Task 1. The task began with the development of a high-level model describing the factors that influence software reliability. This high-level model is based on the principles described in Appendix 7.0A of the NRC's Standard Review Plan for Light Water Reactors.[7] These principles were used to produce two software reliability prediction methods. One method uses a Bayesian Belief Network (BBN) to model prior beliefs about the reliability of a typical software element and then uses information from the process and product measures to update this belief. The other method mathematically combines measures to predict fault density which could then be converted to a reliability prediction.

Both models depend heavily on the opinions of a few researchers as the basis for both their structure and their parameters. Therefore, the accuracy of numerical estimates from these models is highly suspect. The models proposed may, provide a framework within which the influences on software reliability may be debated and data or additional opinions about the precise nature of these influences can be gathered.

The BBN model in particular offers a method by which information from software measures can be combined with NRC review and inspection results to improve the objectivity and repeatability of NRC assessments of software quality.

1.6.1 Considerations

The choice of potential reliability measures, and the evaluation of these measures, was heavily influenced by the special requirements of the NRC. Specifically, the following factors were used in the evaluation:

- NRC reviews of digital instrumentation and control systems submitted by applicants and licensees are expected to be governed by the Standard Review Plan (SRP) Section 7, "Instrumentation and Controls" and the supporting Regulatory Guides referenced by the SRP. Appendix 7.1-C of the SRP notes that the causes of software unreliability are fundamentally different than those addressed by hardware reliability estimation methods. Currently numerical reliability estimation methods do not readily apply to software used under circumstances of very infrequent demand. Consequently, the SRP describes in detail the qualitative method to be used by the NRC in drawing inferences about the reliability of software in digital instrumentation and control safety systems. The SRP emphasizes a three-step approach to the software portion of the reviews. Step 1 requires review of the software planning documents; Step 2 requires review of the applicant/licensee documentation that demonstrates that the plans have been followed; and Step 3 requires that the design outputs resulting from the software development meet specified acceptance criteria. Branch Technical Position (BTP) HICB-14, Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Safety Systems, describes the assessment of software in considerable detail. BTP HICB-14 is basically a qualitative review of the software development process based on current state-of-the-practice. The first goal of this project was to provide a quantitative estimate of software reliability that could be used to support the review process outlined in BTP HICB-14 and SRP Chapter 7.
- Given the NRC emphasis on evaluating submittals early in the lifecycle, a second goal was to evaluate potential measures that might be usable in a PRA developed early in the software lifecycle. The accuracy requirements for the reliability figures required for the software portion of a PRA need be no higher than those used for hardware portions. That is, an order-of-magnitude estimate is considered sufficient.[*] While this goal was a consideration during this study, NRC directed LLNL to focus mainly on the first goal.

[*] For example, the individual component failure rates that made the most important contributions to uncertainty in the ABWR PRA has error factors of between 5 and 15 [General Electric 1993]. This corresponds to a span of 1.4 to 2.3 orders of magnitude at the 95% confidence level.

1.6.2 Task 2 Process

Task 2 consisted of the following steps.

1. Review existing and proposed software reliability prediction methods that utilize measures beyond statistical test results.
2. Develop a high-level model outlining the factors that influence software reliability.
3. Select software reliability prediction methods that fit the high-level model. The Bayesian Belief Model (BBN) and Rome Laboratory combinational models were selected because they appeared to be the most promising techniques discussed in the current literature.
4. Select measures from the Task 1 report for use in the models.
5. Develop the BBN model.
    a) Develop an influence diagram for requirements phase.
    b) Develop an influence diagram showing relationship between phases.
    c) Develop remaining phase models and integrate into overall influence diagram.
    d) Estimate model parameters.
6. Develop the combinatorial model.
    a) Select the model form.
    b) Develop "confidence factors" to fit the selected form.
    c) Develop a method to combine confidence factors into a model.

1.6.3 Summary of Results

Two modeling methods were selected: a Bayesian Belief Network based upon the proposal by Martin, et.al.[8], and a combinatorial model based on Rome Laboratory's "Guidebook for Software Reliability Measurement and Testing"[9].

The BBN models the factors that influence software reliability and the influence of these factors upon the software measures identified by Task 1. As evidence about the software and the software development process is gathered in the form of measures, it is entered into the model to update the reliability prediction. The BBN model developed considers requirements, architectural design, detailed design, implementation, and validation testing. It produces a discretized prediction of the reliability distribution for the software under consideration.

The Rome Laboratory model uses checklists to develop quality factors used to adjust a base failure rate established for the given type of software. The combinatorial model developed for this report is equivalent to the Rome Laboratory model except that the quality factors are estimated based upon the candidate software measures identified by Task 1. The combinatorial model does not explicitly consider different development activity groups. It produces a point estimate of software reliability.

Both models require extensive validation before the numerical predictions they generate could be considered credible. The BBN model, however, appears to provide a method that can be used to combine software measures with NRC inspection results to produce a prediction of software quality. This may offer a way to improve the quality and repeatability of NRC software audits.

If the software engineering community (perhaps in the form of a representative group of experts) can agree on the structure of either model, it could provide a framework for further research to better define the mathematical relationship between the selected measures and software reliability or quality. The research might take the form of data collection from actual software projects, controlled experiments, or formal elicitation of expert opinion.

1.7 Assumptions

Four assumptions were necessary to constrain the models.

a) The software and the development organization under consideration are assumed not to be seriously flawed. That is, an abbreviated review in accordance with the review practice and criteria described in Standard Review Plan Appendix 7.0-A and BTP-14 would not lead to a summary rejection of the software products. The assumption affects the selection of the base reliability prediction for the combinatorial model, the prior beliefs of the BBN model, the practical ranges of the measures considered, and the conditional probability distributions in the BBN model.

b) Initial versions of software products that have very high fault densities have been reworked from scratch and subjected to complete V&V rather than being simply revised with V&V applied only to the changes. This assumption has effects as discussed in the first bullet.

c) No software is perfect. This establishes a performance criteria for the models that they never predict reliability of 1.

d) Safety-related software would not be released for use with known fatal problems, and a minimal amount of validation testing would have been conducted.

2. PRINCIPLES BEHIND MODELS

Both of the models discussed above are built upon the assumptions about the influences on software quality and reliability that are embodied into the NRC review process described in the Standard Review Plan.

2.1 Influences on Software Reliability

Any reliability prediction model should reflect the basic assumptions about what factors influence software reliability. The NRC's Standard Review Plan reflects an accepted philosophy about the relationship between the internal attributes of software development processes and products and the external attributes of software quality and reliability. Consequently, the philosophy embodied in the SRP offers a reasonable starting point in developing a quantitative method for predicting reliability based upon measurement of the internal attributes. In particular, Appendix 7.0-A, Branch Technical Position HICB-14, and Regulatory Guide 1.172 describe NRC's views of what information is needed to form a judgement about software reliability. The SRP process (described below) was developed based on the current state of the art, which is totally process-oriented and judgmental, based on the reviewer's knowledge, and hence is a qualitative measure. The SRP lacks specific quantitative measures of reliability. The main thrust of this study is to develop usable metrics.

Essentially, SRP Section 7.0-A reflects the view that, qualitatively, the reliability of software can be established by confirming that:

1. The software development process was well planned;
2. The planned development process was faithfully implemented;
3. The design outputs (products) meet the software functional requirements; and
4. The design outputs have characteristics consistent with having been developed through a well-planned and well-implemented process.

With the exception of confirming that design outputs meet functional requirements, none of this information alone is sufficient to form a judgement about reliability. Exhaustive review and testing of design outputs may be adequate by itself, but is not practical for either the design or the review organization. Consequently, the judgement about reliability is formed from imperfect knowledge of each of these topics.

Where actual experience with a product is available, the SRP acknowledges that this experience data may be used to supplement what is learned from examination of planning, process, and design outputs.

The evaluation process described in SRP Appendix 7.0-A pre-supposes that the software will be developed by a competent organization using an adequately trained and experienced staff. (This is pre-supposed by the instrumentation and control reviewers because the organizational and personnel factors are examined by other parts of the NRC.)

Finally, the NRC review process encourages that safety systems be simple and recognizes that more extensive review is needed to gain confidence in systems that are more complex.

2.2 Modeling Approaches

After a review of current metrics (see Volume I) we determined that the best approach was via modeling. The overall approach taken was to develop first a high-level reliability model that describes the conditions that influence software reliability. We examined the measures identified in Task 1 to understand what information those measures give us about the conditions that influence reliability. Then we built models that reflect these influences.

Two types of models were developed. The first is a Bayesian Belief Network that models the influences of conditions upon each other as conditional probabilities and uses Bayes rule to propagate the observable conditions to a final estimate of the probability of the not directly observable condition. The second is a combinatorial model based on ROME Laboratory's work, that performs a simple mathematical combination of value of observed measures to provide a prediction about reliability.

2.2.1 Bayesian Belief Networks

A Bayesian Belief Network (BBN) is a graphical network (influence diagram) and associated probability distributions that represent probabilistic relationships among variables, even if the relationships involve uncertainty, unpredictability, or imprecision. The network is made up of nodes and arcs where the nodes represent uncertain events and the arcs the causal/relevance relationships between the events. This network, along with an associated set of node probability tables (NPTs), represents the relationships in the network. BBNs can be used to help make optimal decisions, develop control systems, or make plans and predict events based on partial or uncertain data. Jensen[10] offers an excellent introduction to BBN modeling. A good introduction is also available on the Hugin Expert A/S website.[11]

BBNs combine the advantages of an intuitive visual representation with a sound mathematical basis in Bayesian probability. Although Bayesian probability has been around for some time, the possibility of building and executing realistic models has only recently been made possible through algorithms and software tools, like Hugin and Hugin Lite (the BBN modeling tool we used). With BBNs, it is possible to articulate expert beliefs about the dependencies between different variables and to propagate the impact of evidence on the probabilities of uncertain outcomes, such as future system reliability.

The key feature of BBNs is that they enable us to model and reason about uncertainty. BBNs are a way of describing complex probabilistic reasoning. The advantage of using a BBN is that the BBN represents the structure of the argument in an intuitive, graphical format.

The main use of BBNs is in situations that require statistical inference: in addition to statements about the probabilities of events, the user knows some evidence; that is, some events that have actually been observed, and wishes to infer the probabilities of other events not yet observed, or for which direct observation is impossible.

2.2.1.1 Strengths

Bayesian analysis can be used for both forward and backward inference. The major benefit of Bayesian inference over classical statistical inference (which deals with confidence levels rather than statements of probability) is that it explicitly describes the fact that observation alone cannot predict the probability of unobserved events, without some pre-existing information about the latter. In the Bayesian interpretation, a probability describes the strength of the belief which an observer can justifiably hold that a certain statement of fact is true (subjective probability). The subject, after observing the outcome of an ‘experiment’ (i.e., collecting new data), updates the belief held before the experiment (the ‘prior probability’), producing a posterior probability. The need to assume prior beliefs is a key part of Bayesian inference.

BBNs also offer the advantages that they allow inferences to be based upon a combination of objective and subjective evidence and allow these inferences to be drawn based upon limited input data. As additional data are gained the confidence in the inference drawn will normally increase. Consequently, the technique is very useful in circumstances where the assessors have little control over the specific types of data to be collected about processes and products. Once a set of input data has been collected and analyzed using a BBN model, the model may also be used to conduct sensitivity analyses to identify the additional data that will be most useful in refining the reliability estimate.

2.2.1.2 Weaknesses

Our experience using BBN has shown that the requirement to fill in the relevant Node Probability Tables (NPT) in a sensible way is its weakness as well as its strength. It is not always easy to obtain sensible prior probabilities, even from experts. The derivation of the node dependencies in the network and the form of the network itself needs to be validated.

2.2.2 Combinatorial Model

The Rome Laboratory combinatorial method attempts to identify a mathematical function that relates the observable information to the unknown feature of interest, reliability. Most existing reliability prediction models are of this type. It is a relatively easy way to build models based on various pieces of information. These models may be relatively imprecise, but a well-developed combinatorial model based upon measures may give reliability estimates that are as good as existing models, using more objective and readily available input data.

The construction of a combinational model is a two-step process. The first step toward constructing a combinatorial model was to identify assumptions about the relationship between the observable data and reliability. The second step was to develop simple mathematical functions to express these relationships quantitatively.

The combinatorial method in this project was constructed based on the Rome Laboratory model. Among the existing software reliability prediction models, the Rome Laboratory model was typical and well-known. Its structure was easily adaptable to the use of metrics identified in Task 1 work. It reflected considerable research to determine typical fault densities and transformation of fault density to failure rate. For these reasons, we chose the Rome Laboratory model as the underlying foundation of our combinatorial model.

The general Rome Laboratory model may be expressed as follows:

$$Rp=A*D*S1*S2,$$

where A, D, S1, and S2 represent base reliability, process quality factor, the requirement and design quality factor, and implementation quality factor. In the Rome Laboratory model, these quality factors were chosen from tables or calculated based upon the percentage of checklist questions answered. We proposed, instead, to base these quality factors upon software measures identified in Task 1.

2.2.2.1 Strengths

The combinatorial model has the advantage that it is simpler and easier to understand than the BBN models. The combinatorial model as expressed here also has substantially fewer parameters than the BBN model, so it will be simpler to implement.

2.2.2.2 Weaknesses

A weakness of our Rome Laboratory based model is that it assumes independence between the input parameters. It would be possible to reduce the strength of this assumption, but that would require more complicated formulation and more detailed understanding of the dependencies. The BBN allows the model of dependencies to be built from a combination of simpler, local dependencies.

3. PROPOSED MODELS

Both models are built on the assumptions about the influences on software quality and reliability embodied into the NRC review process described in the Standard Review Plan.

3.1 High-Level Reliability Model

The influences on software reliability discussed in Section 2.1 above can be modeled in the form of an influence diagram as shown in Figure 1. Influence diagrams are directed graphs that show how the states of a system affect other states. For example, Figure 1 shows that the quality of the software development process influences product quality and that product quality influences reliability. Process quality, in turn is influenced by the quality of the development plans and team as well as the complexity of the software. Complexity influences the level of process quality needed to produce a quality product. We used influence diagrams to express the high-level concepts that we believe should be expressed in any reliability prediction model. A number of textbooks including Barlow[12] and Almond[13] provide good discussions of influence diagrams.

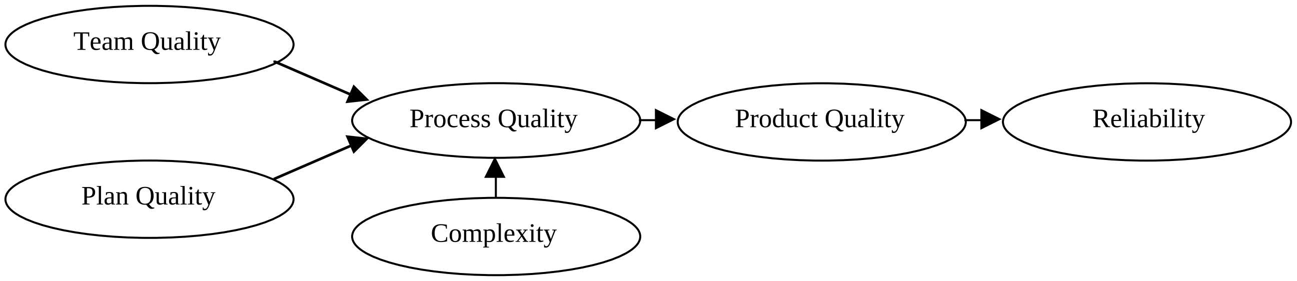


Figure 1. Influence Diagram Model of SRP Assumptions

3.2 Bayesian Belief Network Model

The top level influence diagram of Figure 1 was used to develop influence diagrams for each of the activity groups of the software development lifecycle. The basic steps in developing the BBN model were:

- Create a more detailed influence diagram showing the relationship between process characteristics, product characteristics, and reliability (discussed in Section 4.3.1),
- Modify the influence diagram to describe the relationships between these process and product characteristics and the software measures identified in Task 1 (discussed in Section 4.3.2), and
- Probabilistically describe the influences of process and product characteristics upon the measures and upon each other. (discussed in Section 4.3.3).

Sections 4.3.4 and 4.3.5 discuss the use of the BBN model and provide an example of its use. An evaluation of the model's strengths and weaknesses is provided in Section 4.3.6.

3.2.1 Development of the BBN Model

This typical influence diagram shown in Figure 1 was further refined to create an influence diagram. The software development process actually entailed two independent processes: the development of the software products themselves and the verification and validation (V&V) of these products. To account for the two processes, the influence diagram of Figure 1 was modified as follows.

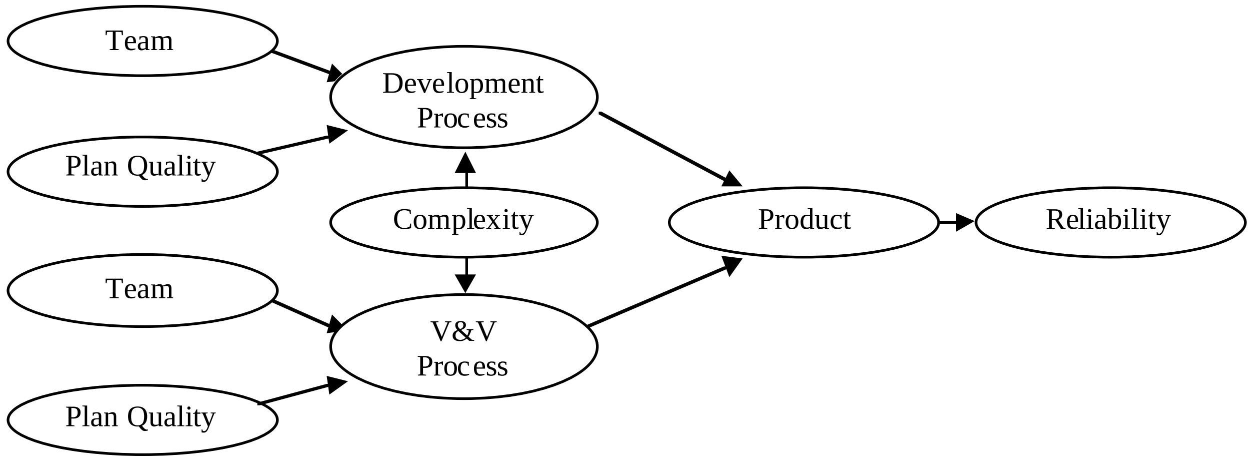


Figure 2. Influence Diagram Showing Separate Development and V&V Processes

The model of Figure 1 represents the state of the software and the development process by which it was produced. The actual state of the nodes shown in Figure 2 is cannot be determined, but it influenced conditions that were observable. The measurements identified in the Task 1 report were observable conditions influenced by the state of the software and the software development process. Figure 2 was, therefore, modified to show the influence on software measures as shown in Figure 3, below. The ovals represent the actual (but unknown) states of the processes and products. The rectangles represent measurements that provide evidence about the state of the process and products.

A further refinement of this model was to recognize that the development process involves producing initial software products. The V&V process evaluates these initial products and identifies errors to the development team. The development team corrects the errors to produce a final product. The model including this refinement is shown in Figure 4.

At this stage, the model lumped all of the development activities together. We improved the model further by using the influence diagram of Figure 4 to separately model the following groups of development lifecycle activities: requirements, architectural design, detailed design, implementation, and testing. Figure 5 shows the model that deals separately with these development activities. Such a “phase-based” model allows the use of more data and provides a mechanism to understand the influence of each phase on reliability. Furthermore, the reliability improvement achieved by error detection in successive phases is at least partially addressed by the back-propagation of later phase evidence through the BBN.

The reliability with which the software performs its safety function is influenced by: 1) the degree to which the software correctly implements the requirements placed upon it by the system in which it operates, and 2) the degree to which the system-level requirements correctly reflects safety requirements. Thus, in developing the phase model of Figure 5 it became clear that it was necessary to include information about the correctness of the original system requirements. This is modeled by the node labeled “System Requirements Quality.”

In the model shown in Figure 5, the requirements, design, and implementation groups were connected together in series. The connections between each of these is a node representing the conditional probability that the safety requirements were correctly implemented if (1) the current phase correctly implemented the requirements imposed by the previous phase, and (2) the requirements from the previous phase correctly reflected the safety requirements.

The validation testing phase plays a fundamentally different role than the development activities and was thus connected into the phase model of Figure 5 differently. Validation testing produces two kinds of outputs: 1) test results, in terms of reliability measures, and 2) test anomolies, assessed and, if necessary, corrected to improve the reliability of the software. The test results provide evidence about the reliability of the software; quality of the test process affect the believability of these results. The feedback of test results into the development is modeled as an anomoly resolution process which increases the reliability of the software more or less depending upon the quality of the testing and the quality of the resolution process.

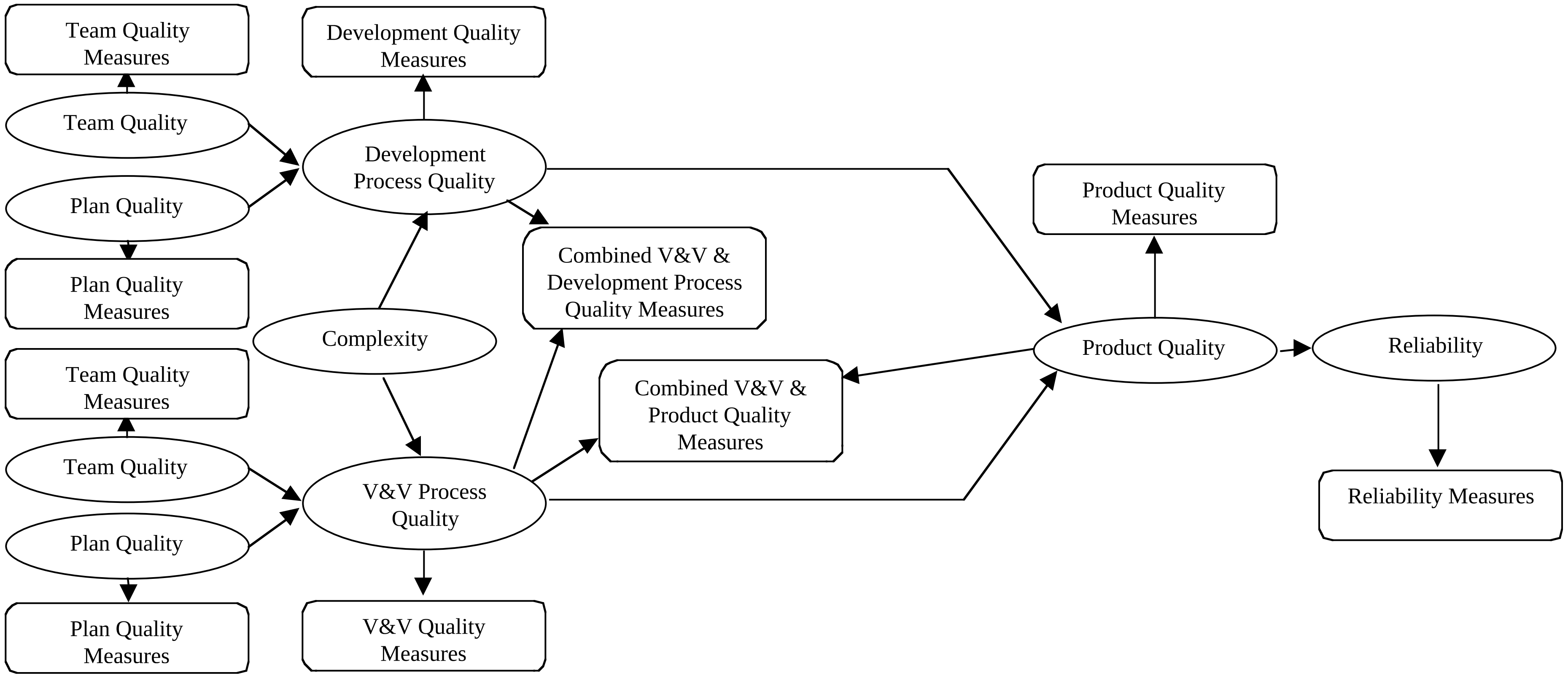


Figure 3. Influence of Process and Product Quality on Observable Measures

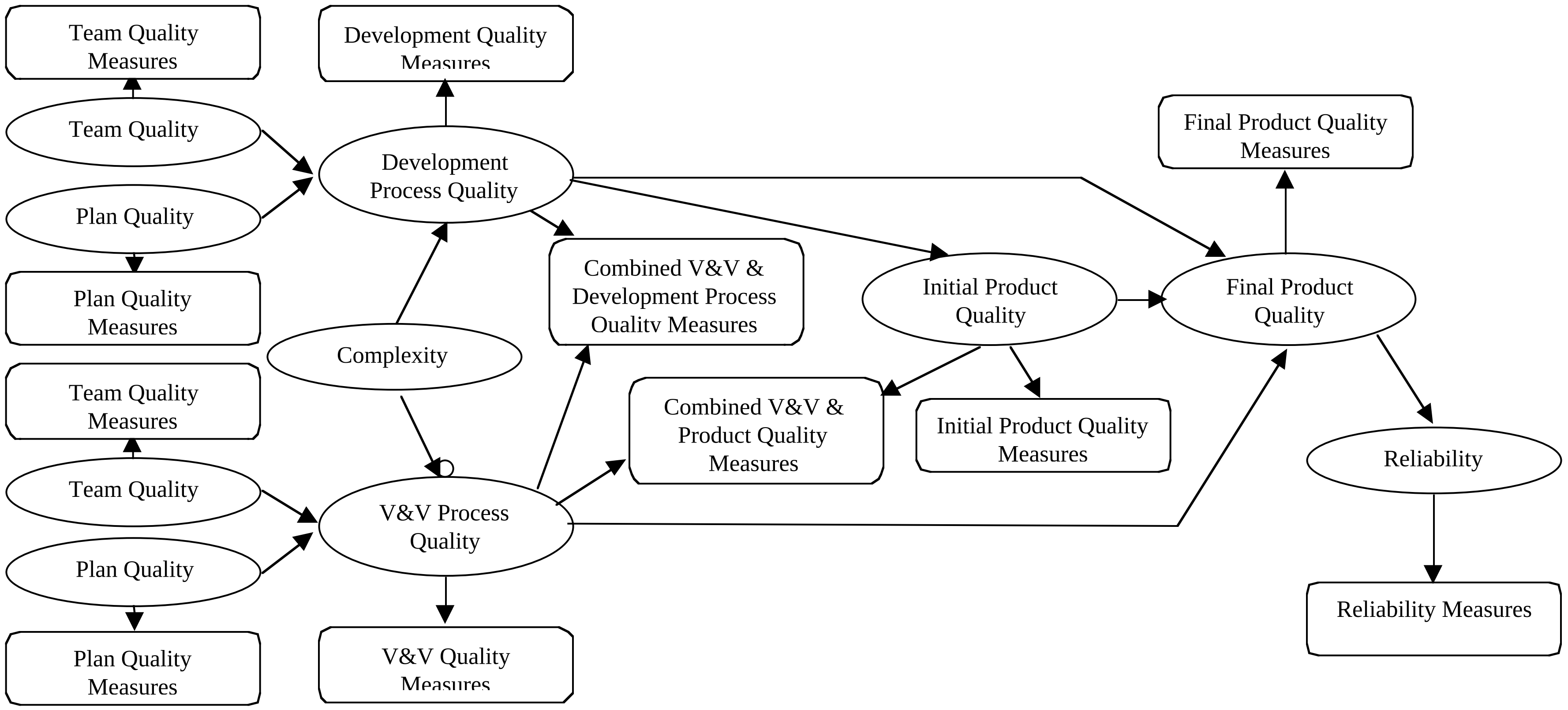


Figure 4. Incorporation of Initial and Final Software Products into the Influence Diagram

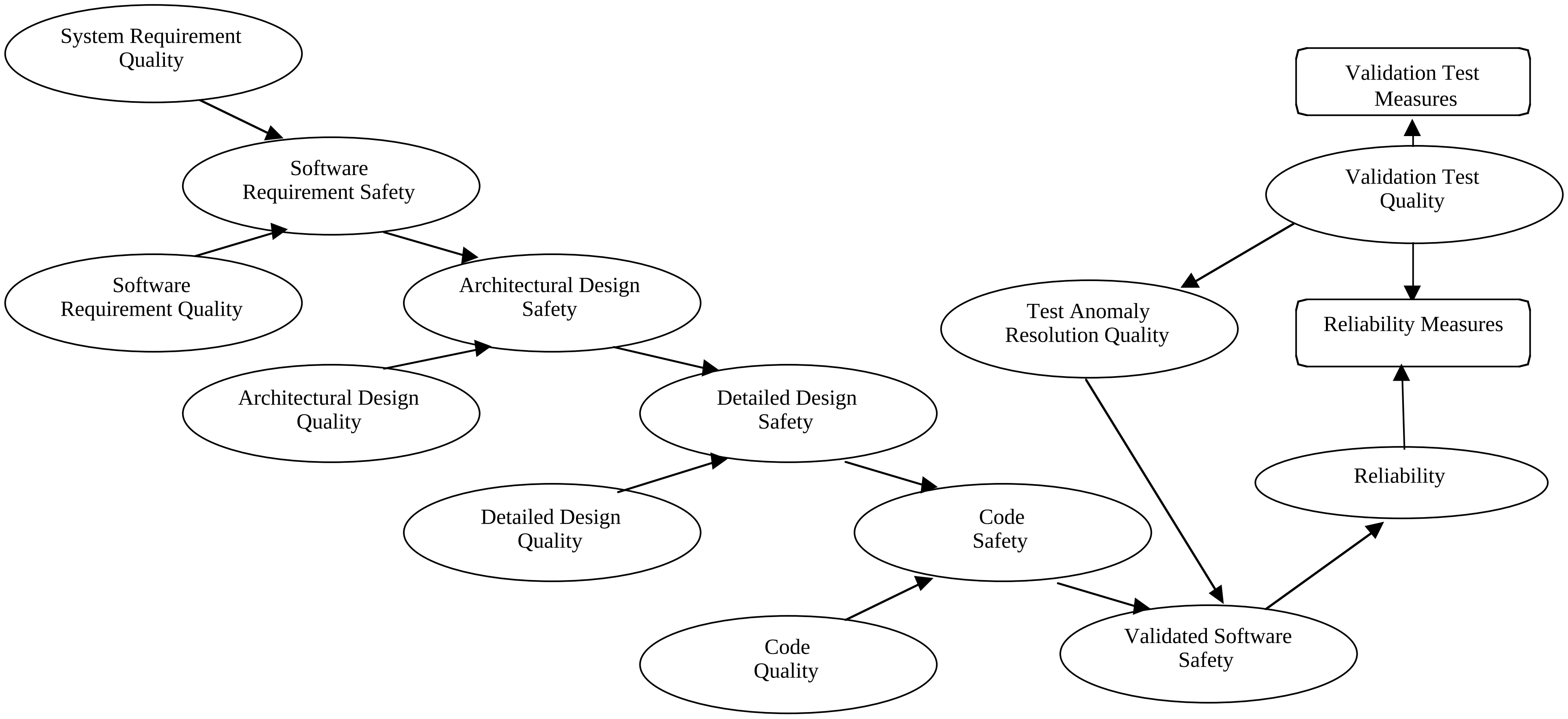

System Requirement Quality
Software Requirement Safety
Software Requirement Quality
Architectural Design Safety
Architectural Design Quality
Detailed Design Safety
Detailed Design Quality
Code Safety
Code Quality
Validated Software Safety
Test Anomaly Resolution Quality
Validation Test Quality
Validation Test Measures
Reliability Measures
Reliability

Figure 5. Phase Model

3.2.2 Incorporation of Measures from Task 1 into the BBN Model

Table 1 shows the highly rated measures for each phase as identified by the Task 1 study [Lawrence 1999]. These measures were examined to determine how to use them as evidence in the BBN model. Table 2 shows the relationship between the measures and the BBN model. In Table 1 the measures are associated with the BBN lifecycle activity in which the measure can be made. Table 2 relates each measure to the activity group about which the measure provides information. These relationships are not always the same. For example, the fault density and defect densities measured during the validation test phase actually provide information about the quality of the design and code. Thus, for use in the BBN model, they are related to the detailed design and implementation phases. The set of measures used in the BBN model is not identical to the set identified in Task 1 as explained below.

Four of the measures considered in Task 1: k-out-of-n model, Markov reliability model, reliability block diagrams, and independent process reliability are not actually measures. Instead, they are methods for building structural models of software to develop a reliability estimate for a larger software element based upon the estimated reliability of the smaller elements of which it is composed. They are, therefore, not included in the BBN model. These modeling techniques were included in the Task 1 study because IEEE 982 treats them as measures. While these techniques are not used in the BBN model, they may be useful in constructing models for use as inputs in the reliability estimates provided by the BBN model.

The Task 1 study treated cyclomatic complexity and minimal unit test coverage determination as different measures because IEEE 982 identifies them thus. They are, however, only different uses of the same measure. Therefore, only cyclomatic complexity is used in the BBN model.

The Task 1 study identified System Performance Reliability as a candidate measure. This measure predicts the probability of meeting timing deadlines. Since the Standard Review Plan expects that the timing performance of safety software will be deterministic, this measure is not used in nuclear power plant safety system applications. Therefore, this measure is not included in the BBN model.

The measure “Fault Number Days” measures the cumulative number of days that faults remain in the software under development. This number may be highly dependent upon the development process schedule and the size of the code. To reduce this dependence, the BBN model uses the measure of the number of phases over which faults remain. This measure is expressed as “Fault Number Phases.” It may be useful to consider expressing this in terms of fault density.

In an attempt to identify additional process measures, the list of measures from Task 1 was re-examined. Task 1 identified the following measures as process measures.

- Cost —The Task 1 study rated it as relatively credible and repeatable, but it has not been validated and there is little experience with its use. It is not included in the BBN model.
- Fault number days — included in the Task 1 list of candidate measures.
- Functional test coverage — this measure was included in the BBN model as an indication of test quality. The Task 1 study rated it relatively highly with respect to all quality criteria.
- Man-hours per major defect detected — Included in the Task 1 list of candidate measures.
- Mean time to discover next k faults — this measure is very similar, but more quantitative than the cumulative failure profile measure identified in Task 1. If directness is ignored as a ranking criterion, this measure is preferable to the cumulative failure profile and thus replaces that measure in the BBN model. For the purpose of the BBN model, k is taken at 1, so that the mean time to failure predicted by the reliability testing is the actual measure used in the model.
- Modular test coverage — included in the Task 1 list of candidate measures.

- RELY – Required software reliability — not added to the measure list because the Task 1 analysis rated this measure low against all of the quality criteria.
- Requirements change requests – this measure was included in the BBN model as an indication of the requirements development process quality. Task 1 ranked it high with respect to credibility and repeatability, and the measure is relatively well validated.
- Schedule —The Task 1 study rated it as relatively credible and repeatable, but it has not been validated and there is little experience with its use. It is not included in the BBN model.
- Software capability maturity model — this measure was included in the BBN model as an indication of development process, V&V process, and test process quality. Task 1 ranked it moderately with respect to credibility and repeatability, but the measure is relatively widely used and validated to some extent.
- Software process capability determination — this measure is similar in purpose to the software capability maturity model, but it is not as repeatable or as well validated. Therefore, it is not included in the BBN model. Preference is given to the software capability maturity model.
- Test accuracy — the Task 1 study ranked this measure relatively low on all quality criteria except repeatability. It is not included in the BBN model.

None of the measures identified as candidates for further study in Task 1 provide information about plan quality or team quality. Furthermore, few of these measures relate to development or V&V process quality. This is due, at least in part, to the Task 1 assumption that preference should be given to direct measures of reliability. The BBN model allows the use of indirect measures in the reliability estimation process; therefore, for this use of the measures, directness is not a useful criterion for ranking.

After the evaluation of all known process measures and addition of measures to the model as discussed above, there is a significant lack of measures that can provide evidence of team or planning quality. Therefore, there is little benefit to including these explicitly in the model. Consequently, the individual phase model of Figure 4 was simplified to lump planning, team and process implementation together as illustrated in Figure 6.

Figures 7 to 11, below, show the complete model, expanding each of the phase quality nodes of Figure 5, using the generic phase model of Figure 6, and incorporating the phase specific measures described in Table 2. The node names in these figures incorporate a letter to identify the associated phase. The meaning of each node is described in more detail in Appendix A.

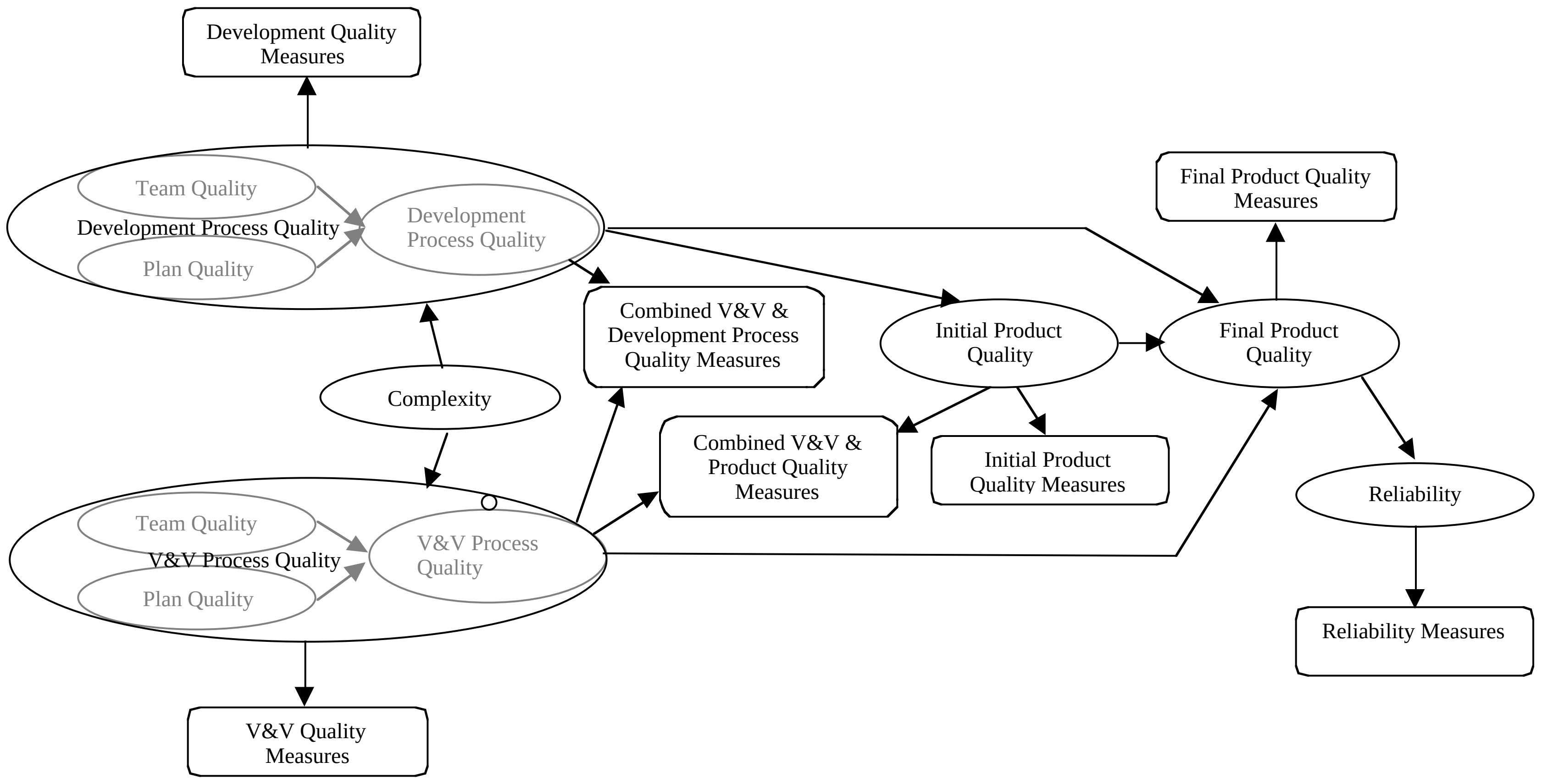

Development Quality Measures
Team Quality
Development Process Quality
Plan Quality
Development Process Quality
Combined V&V & Development Process Quality Measures
Initial Product Quality
Final Product Quality Measures
Final Product Quality
Complexity
Combined V&V & Product Quality Measures
Initial Product Quality Measures
Reliability
Team Quality
V&V Process Quality
Plan Quality
V&V Process Quality
Reliability Measures
V&V Quality Measures

Figure 6. Influence Diagram Model of SRP Beliefs

Table 1. Measures selected for consideration in reliability estimation by the Task 1 study

Lifecycle Phase	Measures Selected
Requirements	Reviews, inspections and walkthroughs Man hours per major defect detected Cause-and-effect graphing Function point analysis Project initiation reliability prediction
Architectural Design	Requirements traceability K-out-of-n model Markov reliability model Reviews, inspections and walkthroughs Graph-theoretic static architecture complexity Reliability block diagrams Man hours per major defect detected Function point analysis
Detailed Design	Design defect density Cyclomatic complexity Independent process reliability K-out-of-n model Markov reliability model Reviews, inspections and walkthroughs Man hours per major defect detected Reliability block diagrams System design complexity
Implementation	Minimal unit test case determination Design defect density Cumulative failure profile Code defect density Cyclomatic complexity Bugs per line of code (Gaffney estimate) Markov reliability model Independent process reliability
Testing	Fault density System performance reliability Design defect density Run reliability Failure rate Cumulative failure profile Reliability growth function Code defect density Modular test coverage
Operation	System performance reliability Run reliability Cumulative failure profile Mean time to failure Reliability prediction for the operational environment

Table 2. Use of measures in the BBN model

Measure	**Requirements**	**Architectural Design**	**Detailed Design**	**Implementation**	**Validation Testing**	**Test Anomaly Resolution**	**Reliability**
Bugs per Line of Code (Gaffney)							Prior estimate of reliability
Cause and Effect Graphing	Initial Product Quality Final Product Quality						
Code Defect Density				Initial Product Quality Final Product Quality	Initial Product Quality Final Product Quality		
Design Defect Density			Initial Product Quality Final Product Quality	Initial Product Quality Final Product Quality	Initial Product Quality Final Product Quality		
Fault Density				Final Product Quality			
Fault Number Phases	Final Product Quality	Final Product Quality	Final Product Quality	Final Product Quality			
Function Point Analysis	Initial Product Quality Final Product Quality	Initial Product Quality Final Product Quality				Initial Product Quality Final Product Quality	
Man-hours per Major Defect	Initial Product Quality V&V Process Quality	Initial Product Quality V&V Process Quality	Initial Product Quality V&V Process Quality	Initial Product Quality V&V Process Quality		Initial Product Quality V&V Process Quality	
Requirements Traceability		Initial Product Quality Final Product Quality					

Measure	**Requirements**	**Architectural Design**	**Detailed Design**	**Implementation**	**Validation Testing**	**Test Anomaly Resolution**	**Reliability**
Failure rate							Observed Reliability
Project Initiation Reliability Prediction							Prior estimate of reliability
Reliability Prediction for the Operational Environment							Prior estimate of reliability
Graph-theoretic static architecture complexity		Complexity				Complexity	
Cyclomatic Complexity			Complexity	Complexity		Complexity	
System Design Complexity			Complexity			Complexity	
Run Reliability							Observed Reliability
Modular Test Coverage					Final Product Quality		
Reliability Growth Function					Final Product Quality		
Reviews Inspections and Walkthroughs	V&V Process Quality	V&V Process Quality	V&V Process Quality	V&V Process Quality	V&V Process Quality		
Mean Time to Discover Next k Faults					Final Product Quality		
Functional Test Coverage					Final Product Quality		
Requirements Change Requests per Requirement	Development Process Quality						
Software CMM	Development Process Quality V&V Process Quality	Development Process Quality V&V Process Quality	Development Process Quality V&V Process Quality	Development Process Quality V&V Process Quality	Development Process Quality V&V Process Quality		

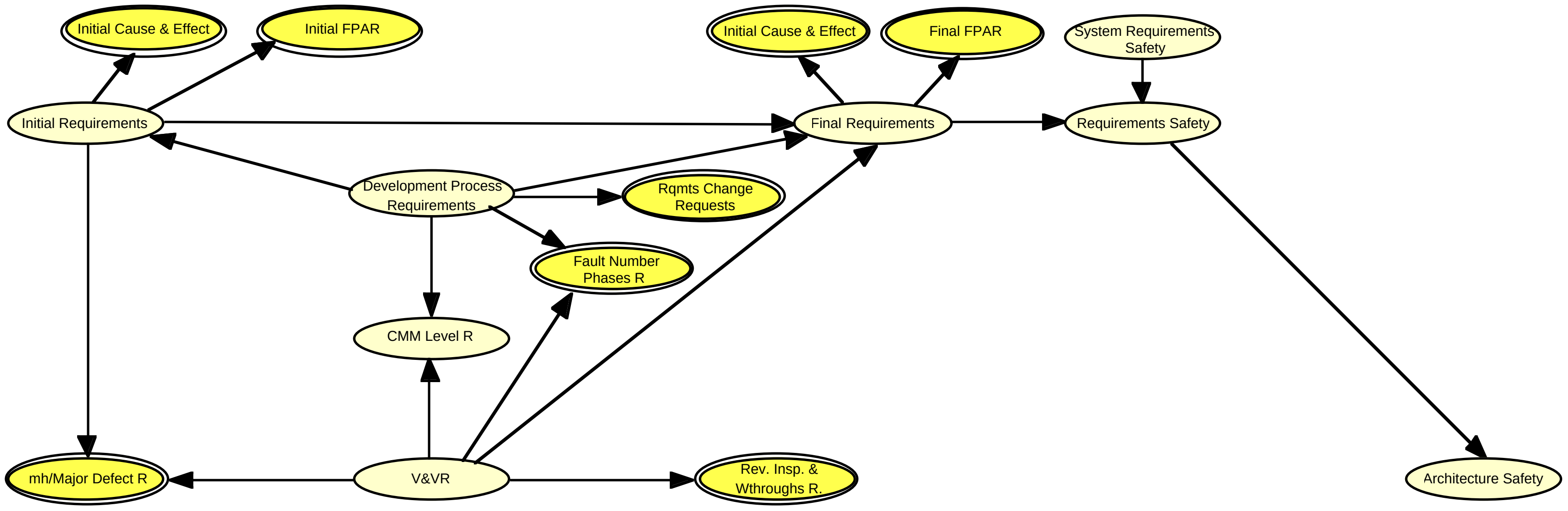


Figure 7. Requirements Phase BBN Model

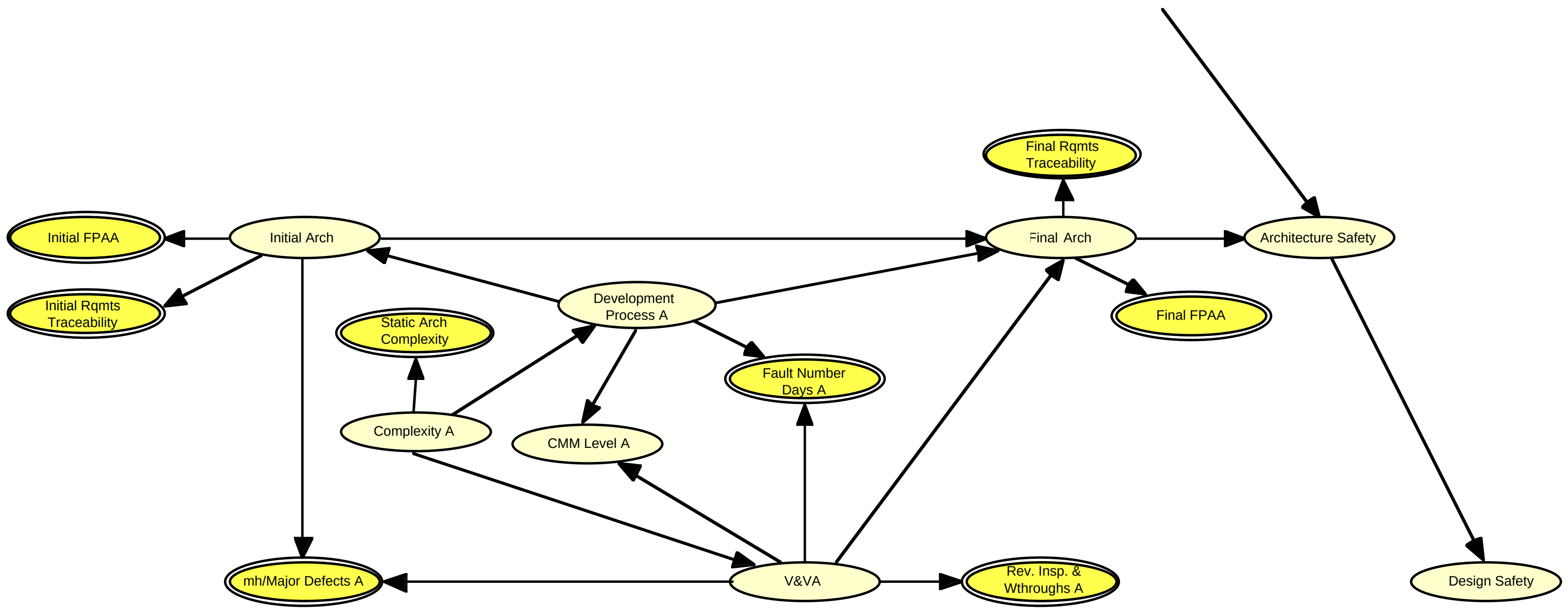


Figure 8. Architecture Phase BBN Model

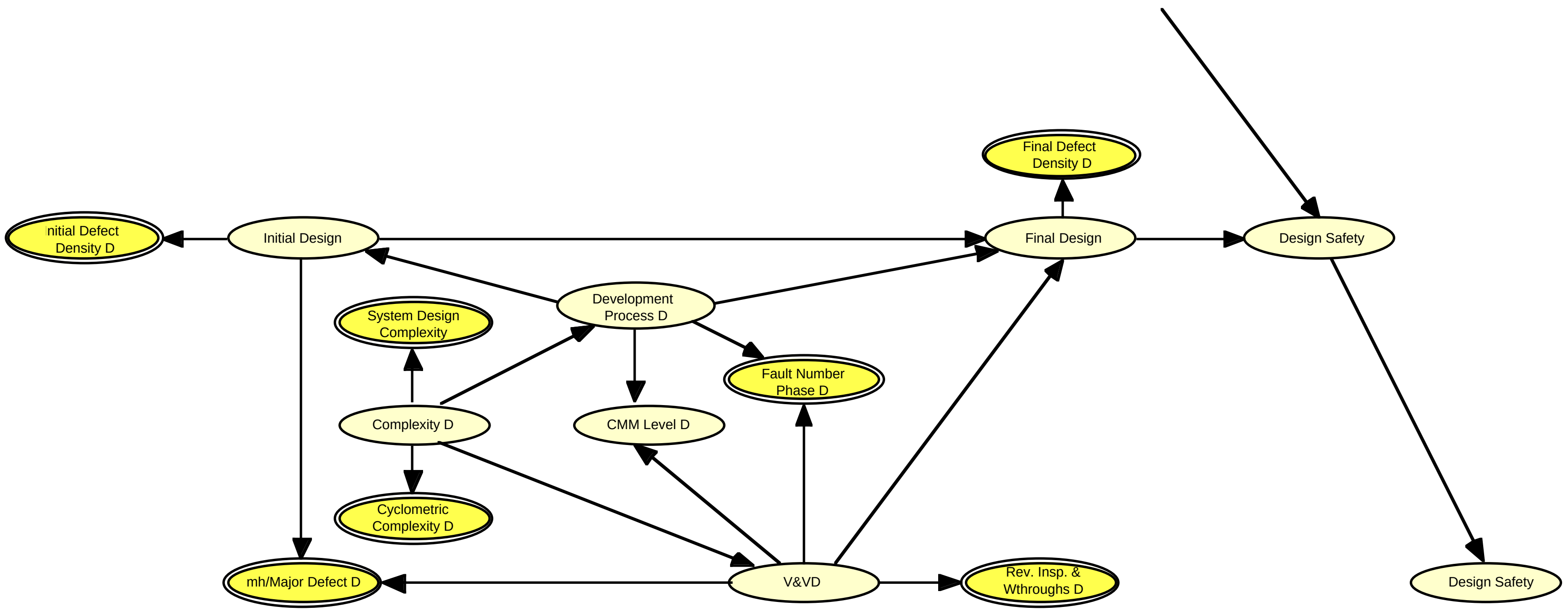


Figure 9. Design Phase BBN Model

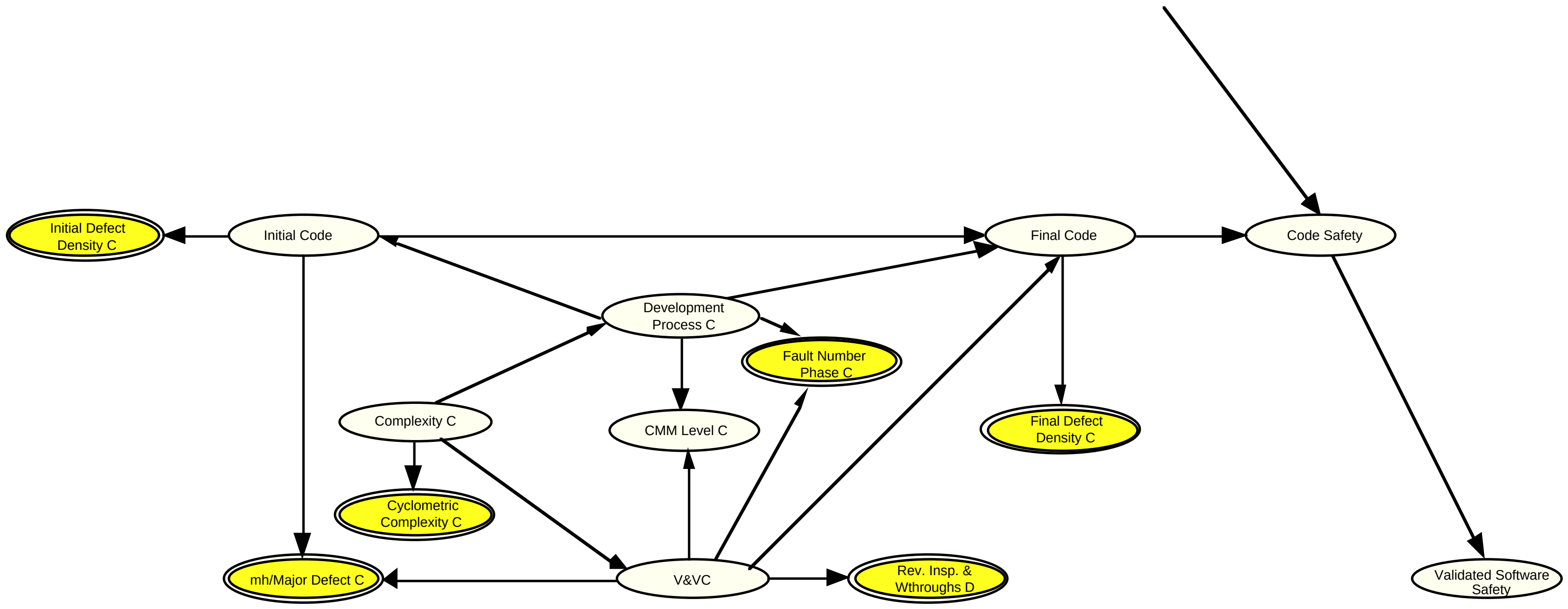


Figure 10. Coding Phase BBN Model

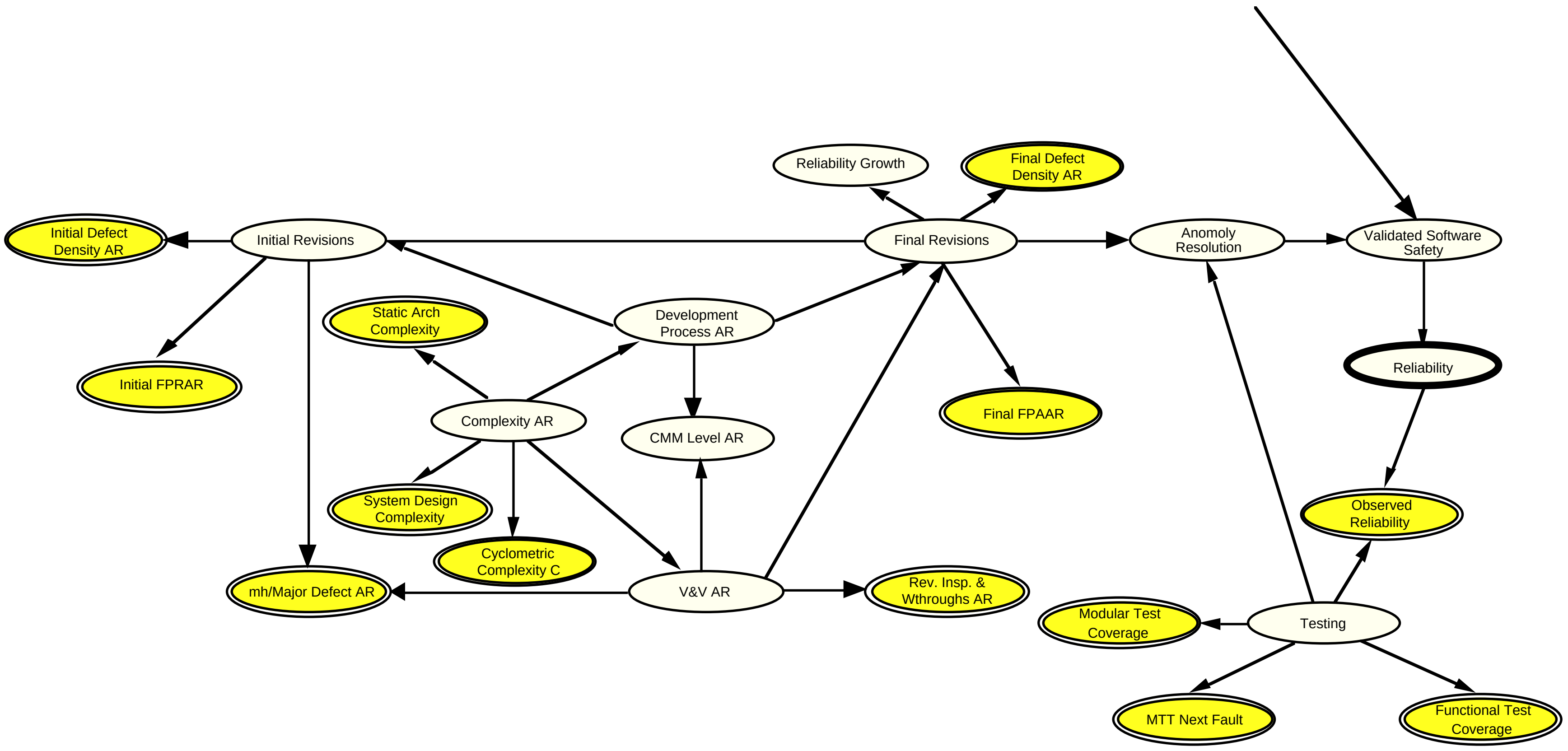


Figure 11. Testing and Anomaly Resolution Phase BBN Model

3.2.3 Estimation of Model Parameters

The parameters that had to be estimated for the BBN model were the probabilities of each node being in a given state conditional upon the state of the other nodes that influence it. The nodes were classified into two types: observable nodes and non-observable nodes. The non-observable nodes represent a condition of the software or software development process about which we would like to know, but cannot directly observe. The observable nodes represent the measurements that could be made on the software or software process. The observations provide evidence which is used to infer the state of the non-observable nodes. The BBN model infers the state of the non-observable nodes based upon the state of the observable nodes (measures), prior beliefs about the states of the non-observable nodes, and conditional probabilities that described the influences of the nodes upon each other.

For nodes that have no input, the prior belief that the node was in a given state must be estimated. For example, with no foreknowledge, one might believe that there is a 90% chance that the developer of safety-critical software will have development processes adequate to develop safe software. This is a prior belief that is modified as evidence about the actual process and products produced is obtained.

For nodes that have inputs, the conditional probability that the node was in a given state (given the state of the influencing nodes) must be estimated. For example, one might believe that a good development process has a 99% chance of producing initial requirements without errors that cannot be detected and corrected by V&V, but that this probability is only 90% for a poor development process.

For the observable nodes, the probability that a given value of the measure is observed (given the state of the related node) needed to be estimated. For example, one might believe that if a developer has a good development process and a good V&V process, there is a 99% chance that the developer's processes will be measured at a CMM level of 3 or above.

Estimation of model parameters required very careful definition of the possible states for each node. Appendix A provides a detailed discussion of each node in the BBN model. The observable nodes were thoroughly defined by the definition of the associated measure. The non-observable nodes were generally defined as follows.

- Development Process Quality: The development process is good enough that it (1) does not introduce safety-significant errors that are undetectable by a good V&V process and (2) can correct detected errors without introducing new safety-significant errors. The implication is that the development process does not introduce errors that are too numerous to be detected and corrected. It may also be important to consider the possibility that the process introduces errors too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. Two states are modeled: true or false.
- V&V Process Quality: The V&V process is good enough to detect all safety-significant errors in a reasonably good initial design output. The implication is that the V&V process can detect all such errors if they are not too numerous. It may also be important to consider the skill of the V&V team with respect to the development team. The V&V team must be capable of thoroughly understanding the development team's products. This is not modeled because there are no metrics to give evidence for the V&V team's skill. Two states are modeled: true or false.
- Complexity: Three states were modeled.

  Very Low – The complexity of the design is such that even a relatively poor development and V&V process are likely to produce a correct product.

  Average – The complexity of the design is such that a good development and V&V process is needed to produce a correct product.

  Very High – The complexity of the design is such that even good development and V&V processes will encounter problems developing a correct product.

- Initial Product Quality: The initial design output is generally correct and complete with respect to the requirements imposed by the previous phase. Any instances in which the previous phase requirements are not met are detectable by a good V&V process and are correctable by a good development process. The implication is that the errors are not too numerous to be detected and corrected. It may also be important to consider the possibility that errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. Two states are modeled: true or false.
- Final Product Quality: The final design output is correct and complete with respect to the requirements imposed by the previous phase. Two states are modeled: true or false.
- Final Product Safety: The final design output is correct and complete with respect to the actual safety requirements (note that these may be different from the documented safety requirements). Two states are modeled: true or false.
- Reliability: The estimated probability of success in response to a safety demand. Four states were modeled:
  1. Reliability < 0.99
  2. $0.99 \leq$ Reliability < 0.999
  3. $0.999 \leq$ Reliability < 0.9999
  4. Reliability ≥ 0.9999

Expert opinion was used to estimate (guess) the probability distribution functions for the model. This is very straightforward for nodes with zero or one input. Only two values must be estimated. For one input node, the two probabilities to be estimated are (1) the probability that the node is true given that the previous node is true, and (2) the probability that the node is true given that the previous node is false. The two other possible conditional probabilities, i.e., the conditional probabilities of the node being false, are complements of the first two.

The probability of the final product quality node is conditional on three inputs: initial product quality, development process quality, and V&V process quality. Eight sets of states (and eight complements) must be estimated. The states were first ordered and then the probability for each state was estimated.

The conditional probabilities for many of the observable nodes were modeled as continuous distribution functions. The BBN modeling tool that we used (Hugin Lite) supports only the use of Normal distributions. In most cases, the distribution of measure values is likely to be more similar to a log normal than normal. For example, if the final requirements document is correct, the probability that the function point analysis measure is very near one should be very high, the probability that it is greater than one must be zero (by definition of the measure), and the probability that it is at a specific value less than one should drop off rapidly as the value of the measure decreases. Such distributions were simulated by a truncated normal distribution.

Hugin uses the mean and variance of the normal distribution as inputs to describe the continuous distributions. It proved difficult to visualize the meaning of the parameters, so a spreadsheet-based tool was developed to assist in estimating the continuous distribution functions. This tool allowed visualization of the functions represented by the mean and standard deviation values input to the model.

Figure 12 shows an example of the tool display. The left pair of charts shows the distribution estimated given that the parent state is true. The top graph shows the estimated cumulative probability that the measure exceeds value x given that the parent state is true. The bottom graph shows the shape of the associated probability density function. The middle pair of graphs show the distributions given that the parent state is false. In this case, the cumulative distribution function shows the probability that the measure is less than x if the parent state is false. The right-hand pair of graphs shows the expected distribution of the metric over all software. This combines the two previous distributions using a prior estimate of the probability that the parent state is true.

Estimators were asked to adjust the mean and variance estimates until the resultant distributions appeared to be reasonable.

Probability Distribution for Initial Function Point Analysis of Initial Requirements

If State is True		If State is False		Expected Distribution of Metric State: unknown
Metric Mean = 0.95	Prior 0.85	Metric Mean = 0.9	Prior 0.15	
Variance = 0.001		Variance = 0.001		Scale 0 to 1

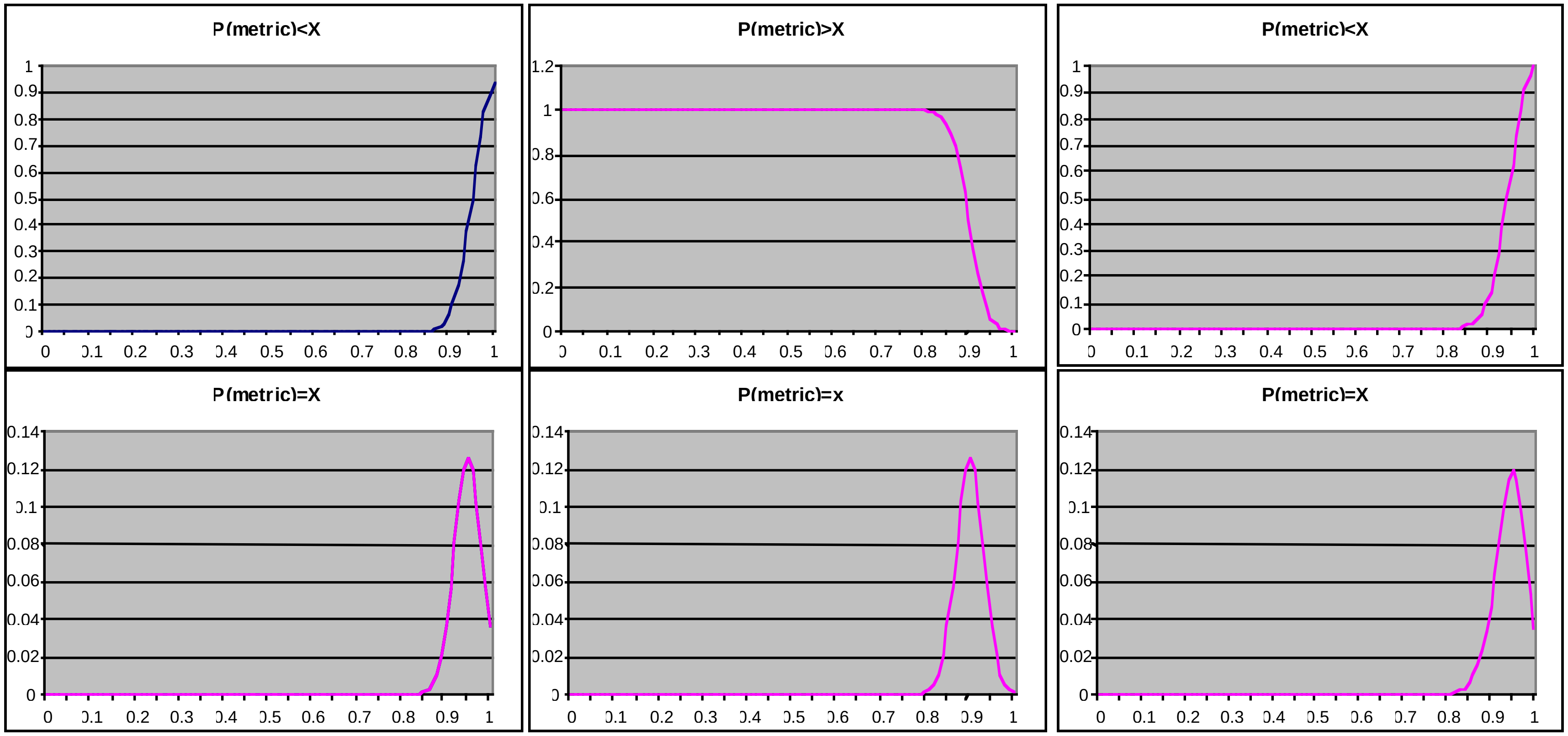


Figure 12. Tool for visualization of continuous distributions

3.2.4 Use of the BBN Model

This section describes how data would be entered into the BBN model to estimate reliability for a specific set of software. The BBN model is used by first changing the prior beliefs to the extent possible to make these particular to the specific application under study. The user should be particularly careful to change the prior beliefs in two nodes:

- **Estimated reliability.** The conditional probability table for this node represents the prior belief about the reliability distribution function for the software. This distribution should be updated based on information from other reliability prediction models such as the Project Initiation Reliability Prediction. It is important to represent both tails of the distribution. Entering a prior probability of zero for any of the reliability ranges will cause the model to always assign zero to the probability that the reliability is within the stated range of reliability.
- **System requirements:** The probability table for this node represents the belief that the requirements which are passed from the system developers to the software developers correctly and completely represent the fundamental safety requirements. All subsequent estimates are heavily dependent upon this estimate. The user should estimate this node based upon insights gained from examination of the system requirements. The default value is a prior belief that there is a 99% chance that the system requirements are correct and complete. A more formal means for estimating this prior belief is beyond the scope of this study.

The user should consider changing the prior beliefs for other nodes which have no parents. These nodes are:

- **Development Process ( ):** These nodes represent the initial belief that the development process is good enough to produce initial phase products that can be perfected through the use of a good V&V process. The default value for these nodes is that there is a 90% chance that the requirements development process is good enough. The user should change this prior estimate if there is information (e.g., previous experience with the development team) leading to the belief that the development process is much better or worse than this. If the user believes that the default estimate is too high, it may well be that the software does not pass the first test for safety related use and estimation of reliability is a moot issue.
- **V&V Process ( ):** These nodes represent the initial belief that the requirements V&V process is good enough to detect all safety-significant errors in the initial product (presuming that the products from the previous phase are correct). The default value for these nodes is that there is a 90% chance that the V&V process is good enough. The user should change this prior estimate similar to the prior estimate of requirements development process quality.
- **Complexity:** There are several nodes that represent complexity of the software architecture, the software design, and the code. These nodes represent the initial belief about the demands placed upon the development and V&V teams by the characteristics of the design and code. Three levels of complexity are modeled: high, medium, low. Low represents a design that is simple enough that success might be expected from even a relatively poor quality development and V&V team. High represents a design that is complex enough that very high quality teams are thought to be needed to have confidence of success. Medium is in between. The default value for the complexity nodes is to assume each of the three levels is equally likely.

The user may also consider changing the prior estimates for any of the other conditional probability tables and the continuous distributions in the model. It is likely, for example, that the user may have some prior judgement about the quality of the development and V&V processes for phases other than the requirements phase. Estimates for other nodes may be changed if the user has organization or project specific information that can be entered.

Once the prior beliefs are changed, the model is run. The initial run displays the expected probability about the state of each node based solely on the prior beliefs entered into the model. For the continuous nodes, the run displays the expected distribution within which the software measures are expected to be found.

As data from measures is obtained, the actual value of the measures is entered into the model and the prediction about software reliability is updated. This use is illustrated in the following section.

3.2.5 BBN Model Evaluation and Comments

The BBN model provides a tool for combining measures of a variety of software characteristics into an overall prediction of software reliability. There are numerous uncertainties, however, that need to be considered when applying the model.

The model is, as all models are, incomplete. Substantial testing would be needed to identify the impact of this and to determine where elements should be added to improve the model and where the model might be simplified without harm. Testing should include validation of the individual phase models and the overall model.

The structure of the model has not been validated. Review by other workers in the field may result in structural changes that improve the model.

The conditional probability estimates that provide the parameters for the model are the opinion of a limited set of researchers. The model may be improved by eliciting opinions from a larger body of researchers. If consensus on the network model could be achieved, however, it could form the framework for experimental and data collection programs designed to replace the expert opinion with measured statistics.

The ability to perform sensitivity analysis is not provided. It would be very useful to know which conditional probabilities and which evidence have the greatest affect on the final result. This information could be used to focus research on improving the model and to provide information about the process and product improvements that are believed to offer the greatest improvement to software reliability. Software could be written to perform a sensitivity analysis on the model. Careful thought would be needed to select the importance measures to be used. Sensitivity analysis will be complicated by the fact that a number of model parameters and input measures interact with each other.

The degree to which the model parameters may be project-specific is unknown. A data collection program as mentioned above would be useful in resolving this uncertainty.

The ability to predict overall software reliability has not been validated. While a single test of the method is planned, limited testing will not be sufficient to provide confidence in the numerical results.

For the most part, these shortcomings are not unique to this BBN model. All existing software (and indeed hardware) reliability prediction methods suffer these problems to some degree or another. The real test for this model will be whether it is a more or less accurate predictor of software reliability when compared to other models.

Regardless of any shortcomings as a reliability predictor, the BBN model represents the process for reasoning about software quality recommended by the Standard Review Plan. The model uses defined software measures as the basis for evaluation rather than the more general acceptance criteria outlined in the SRP's BTP-14. Therefore, the model provides a more transparent, objective, and repeatable method for making decisions about software quality than the review process of BTP-14. BTP-14, however, is strengthened by its consideration for many more attributes of software products and processes than are addressed by the software measures incorporated into the BBN model. There is no reason why findings from reviews conducted using BTP-14 could not be incorporated into the BBN model to provide a quality evaluation tool that combines the strengths of both methods. NRC has developed a draft software review checklist to support evaluations according to BTP-14. Coupling the results of evaluations using the checklist tool as a further source of evidence for the BBN model would be worth consideration as a means to improve tool support for NRC reviewers.

3.3 Combinatorial Model

3.3.1 Introduction to the Rome Laboratory Model

To improve software reliability, Rome Laboratory developed a guide book entitled “Methodology for Software Reliability Prediction.”[14] It is used by Air Force Acquisition Offices to specify achievable and measurable reliability goals in terms of fault density and failure rate and evaluate progress toward those goals at key project milestones. To develop the software reliability prediction and estimation methodology in the guidebook, Rome Laboratory analyzed 59 projects. The result was declared to be useful to any project with high reliability requirements that can be matched to the generic applications used in the guide.

In its software reliability prediction task, the guidebook utilized the following model to predict fault density (Rp) in the source software:

$$Rp = A*D*S1*S2,$$

where, A, D, S1, S2 are input metrics representing initial estimated fault density, the development environment quality factor, the requirement and design quality factor, and the implementation quality factor. At project initiation, reliability is predicted using only the first factor (A). During requirements and design phases, the model user can collect data that allows the first three factors to be used. Data for the fourth parameter becomes available only after coding. The software reliability model is analogous to hardware reliability models developed for MIL-HDK-217[15], in which a base failure rate is established for different use environments and this base failure rate is adjusted by factors that account for differences in materials, design characteristics, stress, and production quality.

For A, the initial estimate of reliability, Rome Laboratory developed an average or baseline fault density by analyzing the observed fault density for a number of projects. Rome Laboratory has estimated this baseline fault density based upon the type of application the subject software represents and has provided methods for converting fault density estimates to reliability estimates. This is used as a starting point for the prediction. The development environment, design, and implementation quality factors can each increase or decrease the reliability estimate by a factor of 2. Thus the Rome Laboratory model can adjust the base reliability estimate over a span of about 1.5 orders of magnitude.

For D, the development environment metric, the development environment is given in the guidebook into three categories: organic, semi-detached, and embedded. The D value is assigned by the user to one of these three categories using the following definitions.

- Organic — Software is being developed by a group that is responsible for the overall application.
- Semi-detached — The software developer has specialized knowledge of the application area, but is not part of the sponsoring organization.
- Embedded — Software that frequently has very tight performance constraints being developed by a specialist software organization that is not directly connected with the application.

For S1, the requirements and design representation metric, the value is given in the guidebook in terms of a product of three factors: S1 = SA*ST*SQ. SA, ST and SQ represent anomaly management, traceability and quality review results respectively.

For S2, the software implementation metric, the value was given in term of a product of four factors: S2 = S*SM*SX*SR. SL, SM, SX and SR represent factors from language, modularity, complexity and standard review.

The S factors are calculated based upon selected metrics, surveys of software development processes, surveys of software products, and the answers to checklist questions posed by the Rome Laboratory method. The quality factors are calculated based upon the number of checklist questions answered in the desired way.

The model is used to estimate fault density in the final software. Fault density can be transformed into a predicted reliability expressed as a failure rate. The Rome Laboratory guide described the following three approaches as ways to perform this transformation:

- Using established empirical values from a table
- Theoretically based transformation function
- Using in-house data to derive an empirical relationship

3.3.2 Combinatory Method Based Upon the Rome Laboratory Model

The Rome Laboratory model was used a basis for a combinatory estimate of software reliability. The proposed model uses a number of assumptions implicit in the Rome Laboratory model and software reliability measures identified in Task 1. The assumptions implicit in the Rome Laboratory model are:

1. The base fault density is known as a result of empirical examination of a number of software projects, or from some other source.
2. Fault density can be converted to failure rate.
3. The type of the development environment and the quality of the design and code for a particular software product can each, independently improve or reduce the base failure rate by a factor of at most 2.
4. The model postulated in this project bases the factors D, S1, and S2 upon the software engineering measures identified in Task 1.

Software reliability measures identified in Task 1 are used for the factors D, S1, and S2. In addition to the assumptions identified above, the following assumptions are made:

1. The value of software engineering measures and fault density can be related by some non-linear function.
2. The relationships between metrics and fault density obey a “law of diminishing returns and diminishing penalties.” That is, at some point, improvements in the values of a metric do not indicate that one should expect a corresponding reduction in fault density and that below some point, degradation in the value of a metric do not indicate that one should expect a corresponding increase in fault density.
3. Very poor results against any one measure indicates that one should expect a relatively high fault density. Correspondingly, all measures must be relatively high in order for the model to calculate relatively low fault densities.
4. Development environment quality is a more useful environment measurement than the development environment type as defined in the Rome Laboratory model.

3.3.3 Incorporation of Measures from Task 1 into the Combinatorial Model

Developing quality factors based upon the measures identified in Task 1 involved the following steps:

Step 1 — Select appropriate measures based on the work done in Task 1.

These measures are: requirements traceability, design defect density, reviews, inspections and walkthroughs, minimal unit test case determination, man-hours per major defect detected, K-out-of-n model, Markov reliability model, cyclomatic complexity, cause & effect graphing, function point analysis, cumulative failure profile, independent process reliability, and code defect density.

As discussed in Section 3.2.2, minimal unit test case determination and cyclomatic complexity are the same measure and so treated. The modeling methods (K-out-of-n and Markov reliability model) are not used because they are not measures. Modular test coverage, the capability maturity model measure, and modular test coverage replaced these deleted measures to improve the representation of process measures in the model.

Step 2 — Develop functions to relate individual metrics to quality factors.

Generally, two kinds of relationships were assumed for the selected measures. For some measures (e.g., man-hours per major defect), higher values indicate better software. For some others (e.g., code defect density), higher values indicate worse software. The functions represent and quantify these relationships between the measures selected and quality. An assumption is that these measures are affected a law of diminishing returns or penalties as described in Section 3.3.2. These relationships are described by an “s”-shaped curve for measures where increases in the measures’ value indicate an increase in quality. A reverse “s” or “z”-shaped curve represents this relationship when decreases in the value of a measure indicate increased quality. During the process to develop these functions, practical bound values must be assigned to the metrics with infinite bounds, based on their characteristics and experience. For example, the measure man-hours per major defect can go from zero to infinity; however, experience has shown that values much greater than 5 are not encountered in successful development processes.

Step 3 — Combine metric quality factor into process and product quality factors that replace the D and S parameters of the Rome Laboratory model.

Although the measures were selected based on ranking, which indicates they have a different priority in suggesting software reliability, they are considered to be in the same order since the ranking indicates only very slight differences.

Step 4 — Convert the defect density to a failure rate using one of the conversion methods described by Rome Laboratory.

3.3.4 Detailed Description of the Combinatorial Model

3.3.4.1 Base Reliability

As described before, the Rome Laboratory model uses an initial reliability estimate based on the application type. The initial reliability estimate might be based upon other measures such as failure rate, reliability prediction for operating environment, or run reliability described in the Task 1 report.

Notice that this combinatory method deals with fault density. Measures results that are not expressed as fault densities (e.g., failure rate measures) require transformation to fault density to use the factors described here.

3.3.4.3 Selection of Measures for Use in the Model

The task 1 study identified the following measures as the most promising inputs for reliability estimation, independent of life-cycle phase: Requirements traceability, design defect density, Reviews, inspections and walkthroughs, Minimal unit test case determination, Man-hours per major defect detected, K-out-of-n model, Markov reliability model, Cyclomatic complexity, Cause & effect graphing, Function point analysis, Cumulative failure profile, Independent process reliability, and Code defect density.

These measures were evaluated for use in the combinatorial model. Some measures proved unsuitable for use because they were duplicates of other measures, some proved unsuitable because they are not actually measures, some needed to be modified to reduce dependency on factors that are not directly related to quality (e.g., program size). A relationship also had to be established between the measures identified and the model’s factors for development process quality, requirements and design process quality, and implementation process quality. Some measures relate to more than one of these.

Once this was completed, it was noted that development process measures were not adequately represented. Therefore, the list of measures identified in task 1 was re-examined to identify measures that could be used in the combinatorial model. Table 3 shows the measures used in the combinatorial model and the disposition of the measures identified by the task 1 study.

Table 3 Measure Use in the Combinatorial Model

Task 1 Measure	**Measure as Used in Combinatorial Model**	**Use**	**Notes**
Requirements traceability	Used directly	Indication of requirements and design quality	
Design defect density	Used directly	Indication of requirements and design quality	
Reviews, inspections and walkthroughs	Used directly	Indication of development process quality	
Minimal unit test case determination	Not used		This measure is the same as cyclomatic complexity, which is used in the model.
Man-hours per major defect detected	Two forms of the measure are used: Man-hours per major defect in requirements and design. Man-hours per major defect in implementation.	Indication of development process quality Indication of requirements and design quality Indication of implementation quality	As a indication of development process quality a lower number indicates higher quality. As an indication of requirements and design or implementation quality a higher number indicates higher quality.
K-out-of-n model	Not used		This item is not used in the model, it is a modeling technique, not a measure
Markov reliability model	Not used		This item is not used in the model, it is a modeling technique, not a measure

Task 1 Measure	**Measure as Used in Combinatorial Model**	**Use**	**Notes**
Cyclomatic complexity	Used directly	Indication of implementation quality	
Cause & effect graphing	Used directly	Indication of requirements and design quality	
Function point analysis	Two forms of the measure are used: Function point analysis for requirements Function point analysis for design	Indication of requirements and design quality	
Cumulative failure profile	Mean Time to Next Failure is used instead	Indication of development process quality	
Independent process reliability	Not used		This item is not included in the model, it is a modeling technique, not a measure
Code defect density	Used directly	Indication of implementation quality	
Added measure	Capability Maturity Model	Indication of development process quality	
Added measure	Requirements Change Requests per Requirement	Indication of development process quality	Original measure is modified to be a change density in order to reduce influence of requirements complexity.
Added measure	Fault number phases	Indication of development process quality	Original measure is modified to be a change time parameter from days to phases to reduce influence of project specific scheduling.

Task 1 Measure	**Measure as Used in Combinatorial Model**	**Use**	**Notes**
Added measure	Graph-theoretic static architecture	Indication of development process quality	Added to include consideration of architectural complexity
Added measure	Modular test coverage	Indication of development process quality	

3.3.4.2 Transformation of Software Measures to Quality Factors

Two functional shapes, "s" and "z" are used to convert the task 1 measures into quality factors. The "s" shaped function is then defined by a linear model defined by 4 points:

1 The lower practical bound of the measure (x_0),

2 The upper practical bound of the measure (x_h),

3 A upper "inflection point" (x_u) the value of the measure above which quality is believed to be very high and further improvements in the measure are believed to indicate only a small improvement in quality.

4 A lower "inflection point" (x_l) the value of the measure below which quality is believed to be very low and further reductions in the measure are believed to indicate only a small decrease in quality.

The inflection points are taken be the points above which increase measures to the maximum or minimum practical value only indicate a 5% improvement or 5% reduction in quality.

The "z" shape function is similarly defined except that the upper inflection point is the point relates to lower quality and the lower inflection point relates to high quality. The two quality functions are then defined as follows.

s-function	z-function
$f = 0.05(x-x_0)/x_l$ for $x \le x_l$	$f = 1 - 0.5\ (x-x_0)/x_l$ for $x \le x_l$
$f = 0.05 + 0.9\ (x-x_l)/(x_u-x_l)$ for $x_l<x<x_u$	$f = 0.95 - 0.9\ (x-x_l)/(x_u-x_l)$ for $x_l<x<x_u$
$f = 0.95 + 0.05\ (x-x_u)/(x_h-x_u)$ for $x \ge x_u$	$f = 0.05 - 0.05\ (x-x_u)/(x_h-x_u)$ for $x_u \ge x$

Consideration of two measures, code defect density and man-hours per major defect illustrate the definition of the "s" and "z" curves.

Longer times to detect defects tend to indicate higher quality of software products when the man-hours per major defect detected measure is used as an indication of product quality. Therefore, the relationship between this measure and quality is modeled as a "s" function. Experience has shown that this measure typically falls between 3 and 5 man-hours for major defect detected, and that a value of less than 3 indicates very poor product quality[17] [IEEE982.1]. The defining points for this function are therefore taken as follows.

Lower practical bound– 0 mh/major defect; for very poor products error detection can be very rapid.

Upper practical bound– 10 mh/major defect; this is taken as twice the maximum normally encountered.

Upper inflection point — 5 mh/major defect; based upon experience that values greater than this either indicate very good products or very poor V&V processes

Lower inflection point — 3 mh/major defect; based upon experience that values less than this indicate poor products.

The resulting quality function is shown in Figure 13.

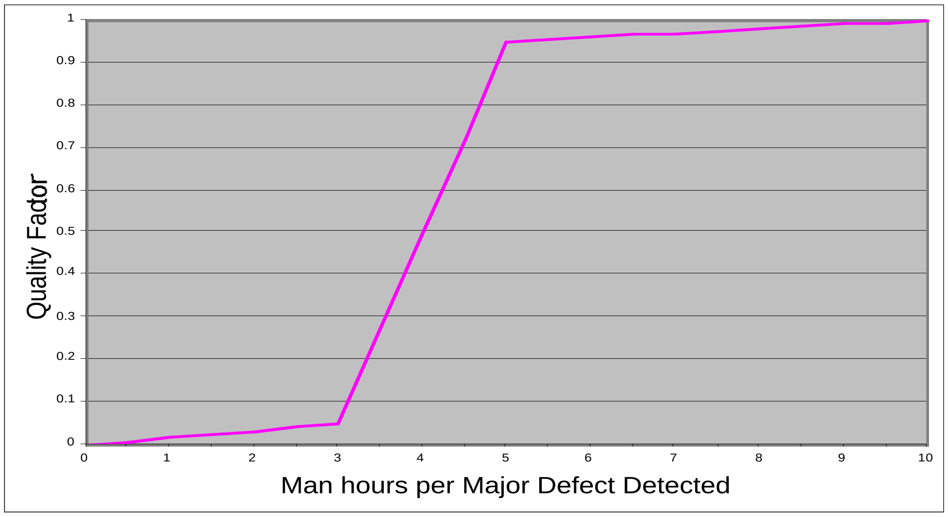


Figure 13. Quality function

Lower than expected defect densities indicate higher quality of software products. Therefore, the relationship between this measure and quality is modeled as a "z" function. Experience has shown that the industry average defect density is approximately 5 defects per thousand lines of source code. The defining points for this function are therefore taken as follows.

Lower practical bound– 0 defects/klosc; zero may be attainable for some very simple software.

Upper practical bound– 20 defects/klosc; this is taken as four times the industry average. Values above this probably indicate that the development team and development process is unsuitable for the creation of safety critical software.

Upper inflection point — 10 defects/klosc, this is taken as twice the industry average and all other factors being equal should result in a predicted defect rate of approximately two times the base.

Lower inflection point — 2.5 defects/klosc, this is taken as half the industry average and all other factors being equal should result in a predicted defect rate of approximately one-half of the base.

The resulting quality function is shown in Figure 14.

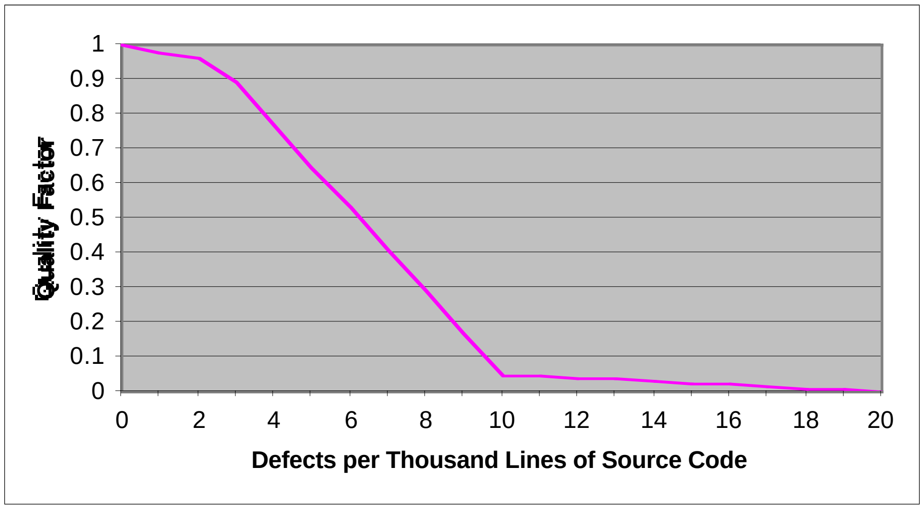


Figure 14. Quality function

Appendix B gives the quality function parameters for each measure.

3.3.4.3 Transformation of Quality Factors into Model Parameters

Three parameter of the Rome Laboratory model are to be calculated:

D — The development environment quality factor,

S1 — The requirements and design quality factor, and

S2 — The implementation quality factor.

For consistency with the Rome Laboratory model the function used to combine quality factors into a single parameter must equal 2 when all of the quality factors equal 0 an must equal 0.5 when all of the quality factors equal 1. Furthermore, all quality factors should be near 1 in order for the result to approach 0.5, and any quality factor being near 0 should cause the result to be near two. This is a restatement of assumption number 3 in Section 3.3.2.

A combination equation that fits these criteria is:

$$X = \frac{2}{2^{2\prod_{i=1}^{n} Q_i^{1/n}}}$$

Q_i are the individual quality factors related to a given model parameter and n is the number of quality measures considered in the parameters. When any quality factors is 0 this reduces to X=2 and when all quality factors are 1, $X=2/2^2 = 0.5$. Figure 15 illustrates the shape of this function for the combination of two quality factors.

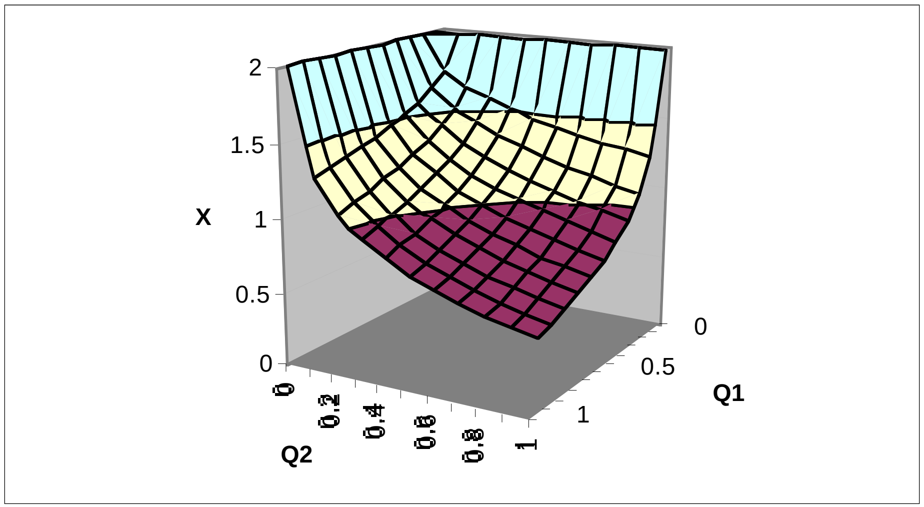


Figure 15. Shape of function for combination of two quality factors

For the calculation of D there are 8 Q_i for the calculation of S1 there are 7 and for the calculation of S2 there are 3. Appendix B gives the specific components of this calculation.

Figure 16 illustrates the combination of the “z”- and “s”-shaped curves above using this method. This would be the function for the S2 parameter, for example, if only two quality factors (man-hours per major defect and defect density) are used.

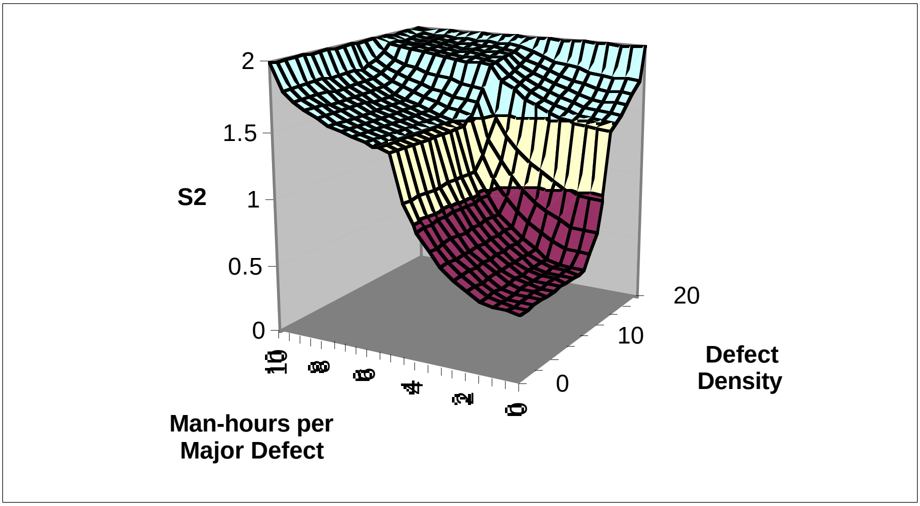


Figure 16. Combination of the “Z”- and “S”-shaped curves

3.3.4.4 Transformation of Fault Density to Failure Rate

As the initial reliability in this model is given in fault density, the result needs to be transformed into failure rate to be used in more sophisticated models like system block model.

Three transformation methods are given in the Rome Laboratory guidebook. One method uses a transformation ratio based on different application types. The guide gives the transformation ratio of 3.8 for process control software which is most representative of nuclear power plant applications.

Then the predicted software reliability in failure rate can be obtained as:

$$\text{Rpf} = \text{Rp} \times 3.8$$

4. TEST PLAN

Both models require extensive testing to validate their suitability for predicting software reliability. The initial project plan envisioned a single test of the reliability models produced by this project. A single validation test cannot prove model validity, but it could give confidence that the models are not invalid. Any validation test case needs to have the following characteristics.

- The software owner must be willing to make information available for this project.
- Information available must include operational reliability data.
- Information on all or most of the software engineering measures used in the model must be available or reproducible from inspection of the available information.
- The owner must consider the information provided to be non-proprietary so that others can reproduce the testing.
- The information must be available within the schedule constraints of the project.

LLNL explored available test cases for more than two years and had no success locating a case that met all of the above criteria. One test case is available using an LLNL proprietary engineering code. Reasonably good information is available regarding requirements and validation test phases; operational reliability is available for several tens of thousands of runs, but little information is available for the design and coding phases. Nevertheless, both methods would be used to predict software reliability; the prediction would be compared to actual experience.

Validation will also be conducted by confirming that both the BBN and combinatorial models behave as expected for variation of individual measures. The general acceptance criteria for this test are:

1. Initial estimate should be between .99 and .999. This represents a range of reliability assertions that we would be willing to accept with very little evidence.
2. No change to any single measure should cause a large change to the prior belief in reliability or the prior belief in the quality of a given phase.
3. The variation of any given measure should produce a reasonable variation in the probability of the states for the daughter node, the probability of the states for the quality of the final product for the associated lifecycle phase, and the overall reliability prediction.
4. Setting all measures to the good should not cause the model to predict perfect reliability nor perfect quality for a given phase. A fundamental belief is that no human design system will ever be perfect.
5. Setting all measures to the bad should not produce a reliability prediction much less than 0.99. Release of software much less reliable than this seems incredible.

Conformance with the first criterion would be determined when the model is first run. Each measure would be varied individually to confirm that criteria 2 and 3 are met. All measures would be varied together to confirm compliance with criteria 4 and 5.

More complete testing would involve testing the models to evaluate simultaneous variations of multiple parameters. Such testing is beyond the scope of this project.

5. CONCLUSIONS

Two possible reliability prediction models were developed using the candidate measures that Task 1 identified as plausible indicators of software reliability. Both models depend upon knowing the relationships between the selected software measures and software reliability. These relationships are unknown. In the absence of details about these relationships the proposed models reflect the thought processes behind accepted methods for subjectively assessing software quality. The measures that were identified by task 1 are used to estimate the software process and product attributes considered in these subjective assessments. This process produced models that produce plausible estimates of software reliability that behave as expected as input data are varied. While the models are not yet useful for making numerical estimates of reliability, they might offer a method for comparing the reliability of different software elements.

A more credible model of these relationships could be developed if sufficient experience data were available to support description of the relationship between reliability and the measures. Previous efforts to collect such data have been unsuccessful largely because developer organizations generally do not collect the information that is needed, and when they do, are often reluctant to release detailed information about failures and internal measures. Furthermore, measures are not always used consistently from organization to organization. Consequently, collection of data to populate these models would require careful operational definition of the measures.

Further development of the BBN model would require developing a method to determine the state of the non-observable modes independent of the measures used in the model. Specific examples could then be studied to probabilistically describe the relationship between the non-observable nodes and the software measures represented by observable nodes. This would allow the development of probably distributions describing the node relationships. It is also possible that such an effort would also reveal the need for modifications to the influence diagram structure. Studying each node in-turn would result in a more defendable model. One might start with one phase and build the model up by a series of data collection and testing on a phase-by-phase basis. Such a study would require collecting information for approximately 200 probability distributions. Validation of the model would require use on many tens of test cases.

Although neither model should be considered a credible method for reliability prediction at this time, they both offer possible approaches that could be further developed. Of the two, the BBN approach appears to be the most promising because it expresses the relationships between the measures and reliability in terms of simpler relationships that are more amenable to study and discussion. Further development of the BBN model should involve developing a wider consensus on the structure and content of the model, followed by determination of the conditional probability distributions for the eventual structure. Developing these relationships from experience or test data may not be practical as discussed above; however, more thorough and structured expert elicitation of the distributions may be sufficient to develop a useable model.

Although many obstacles remain in the path of developing these methods into a reliability prediction method, the BBN approach appears to offer great promise as a quality assessment tool. The BBN model presented in this report essentially models the thought process of an auditor applying the review guidance of the Standard Review Plan. Modeling the process with a BBN makes the evaluation process more transparent and repeatable. Transparency comes in the form of explicitly stating and quantifying the relationships between the reviewers observations and conclusions. This opens up the thought process and detailed judgements to review, criticism, and debate which will lead eventually to improvement of the process. It would also allow maintaining records of review data, review conclusions, and computer system reliability that over time could be used to base the modeled relationships on data rather than on judgement.

The BBN model also offers a method for combining judgements from qualitative assessments (e.g., review according to BTP-14) with quantitative measures to improve the NRC software review process. This path appears to be a viable route to developing assessment tools that can be useful to the NRC staff.

REFERENCES

[1] Institute of Electrical and Electronics Engineers, "IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE 982.1, 1988.

[2] Institute of Electrical and Electronics Engineers, "IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE 982.2, 1988.

[3] Lawrence, J. Dennis, "Assessment of Software Reliability Measurement Methods," Lawrence Livermore National Laboratory, UCRL-ID-136035, 1999.

[4] U.S. Nuclear Regulatory Commission, "Criteria for Digital Computers in Safety Systems of Nuclear Power Plants," Reg. Guide 1.152, Office of Nuclear Regulatory Research, Revision 1, January 1996.

[5] Institute of Electrical and Electronics Engineers, "Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations," IEEE Std 7-4.3.2-1993.

[6] Butler, R., and Finelli, G., "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," IEEE Transactions on Software Engineering, January, 1993.

[7] U.S. Nuclear Regulatory Commission, "Standard Review Plan for the Review of Safety Analysis Reports for Nuclear Power Plants," Chapter 7, Rev. 4, NUREG-0800, June 1997.

[8] Neil, M., Littlewood, B., Fenton, N., "Applying Bayesian Belief Networks to System Dependability Assessments," Proceedings of Safety Critical Systems Club, February 1996.

[9] Rome Laboratory, "Software Reliability Measurement and Testing—Guidebook for Software Reliability Measurement and Testing," HL-TR-92-52, April 1992.

[10] Jensen, Finn, "An Introduction to Bayesian Networks," UCL Press, May 1996.

[11] www.hugin.dk

[12] Barlow, R., "Engineering Reliability," Society for Industrial & Applied Mathematics, April 1998.

[13] Almond, R., "Graphical Belief Modeling," Chapman & Hall, 1995.

[14] Rome Laboratory, "Methodology for Software Reliability Prediction," RADC-TR-87-171, November 1987.

[15] U.S. Department of Defense, "Reliability Prediction of Electronic Equipment, MIL-HDBK-217, February 1995.

[16] Institute of Electrical and Electronics Engineers, "IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE 982.2, 1988.

[17] Institute of Electrical and Electronics Engineers, "IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE 982.1, 1988.

Appendix A

APPENDIX A
DETAILS OF THE BAYESIAN BELIEF NETWORK MODEL

Requirements Phase Model

Name:	**System Requirements Safety**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	System Requirements Definition
Meaning:	Quality of the system requirements with respect to the fundamental plant safety requirements. The node models two states:
1)	Good – All fundamental plant safety requirements are completely and correctly stated in the system requirements document.
2)	Poor – One or more fundamental plant safety requirements omitted or incorrectly stated in the requirements document.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences safety of the software requirements
Comments:	This node models the influence of errors outside of the software development process upon software correctness. It accounts for the fact that software that is correct with respect to the input requirements may still not be incorrect with respect to the actual requirements. The prior estimate should be based upon review of the quality of the system development process, the design bases, and the specifications for integrated hardware / software components.
Reference:	IEEE 982

Probability Table

Correct	0.99
Incorrect	0.01

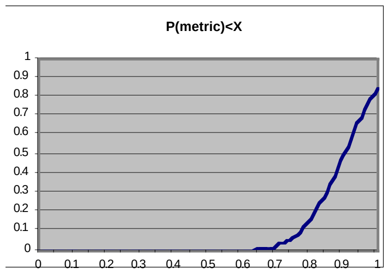
Name:	**Requirements Safety**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Requirements
Meaning:	Quality of the requirements with respect to the fundamental plant safety requirements. The node models two states: 1) Good – All fundamental plant safety requirements are completely, and correctly stated in the requirements document. 2) Poor – One or more fundamental plant safety requirements omitted or incorrectly stated in the requirements document.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences safety of the software design.
Comments:	
Reference:	

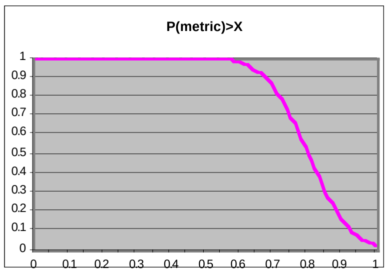
Probability Table

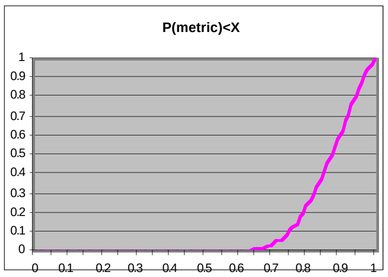
System Requirements Safety	Correct		Incorrect	
Final Requirements	Good	Poor	Good	Poor
Safe	1	0	0	0
Not Safe	0	0	0	0

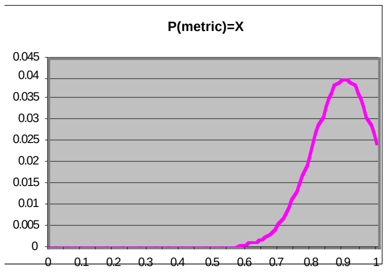
Name:	**Initial FPA R**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Function point analysis quality measure on initial draft of requirements. Measures 1 minus defect density per function point. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the requirements.
Modeled Range:	FPA 0.8 to 1. An initial FPA < 0.8 probably indicates a failed development process.
Use:	Evidence of the quality of the initial requirements.
Comments	
Reference:	IEEE 982

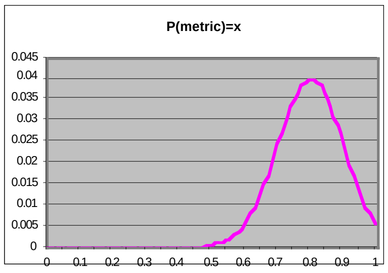
Probability Distribution for Cause & Effect Analysis of Initial Requirements

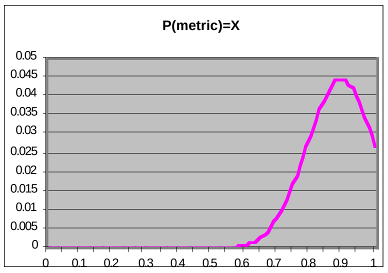












Name:	**Initial Requirements**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the initial requirements. The node models two states: 1) Good – All errors in the initial requirements are detectable by a good V&V process and are correctable by a good development process. The implication is that the requirements errors are not too numerous to be detected and corrected. It may also be important to consider the possibility that errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. 2) Poor – The initial requirements contain some errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1. The probability of the initial requirements being in these states is conditioned by the quality of the development process.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the quality of the final requirements. Evidence of the quality of the development process.
Comments:	
Reference:	IEEE 982

Probability Table

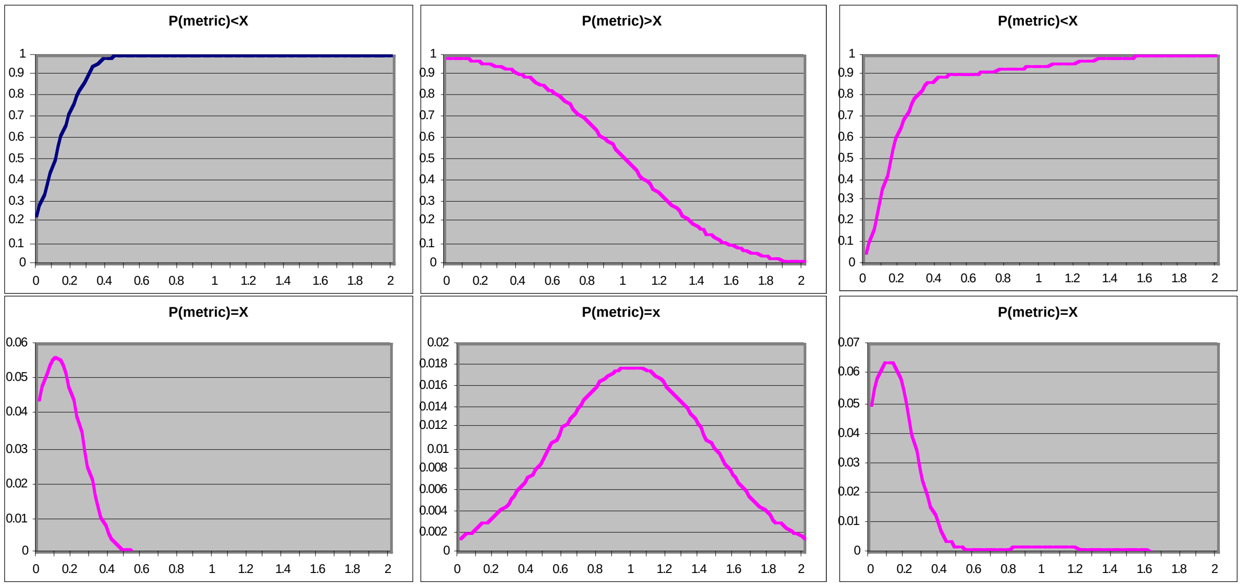
Development Process	Good	Poor
Good	0.99	0.9
Poor	0.01	0.1

Name:	**Development Process R**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the requirements development process. The node models two states: 1) Good – The requirements development process is good enough that it does not introduce safety-significant errors that cannot be detected by a good V&V process and that it can correct detected errors without introducing new safety-significant errors. The implication is that the development process does not introduce requirements errors that are too numerous to be detected and corrected. It may also be important to consider the possibility that the process introduces errors that are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. 2) Poor – The requirements development process is expected to produce errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the quality of the initial and final requirements.
Comments:	
Reference:	

Probability Table

Good	0.9
Poor	0.1

Name:	**Requirements Change Requests**
Type:	Continuous
Observable:	yes
Phase where Measurable:	All phases
Phase that it tells about:	Requirements
Meaning:	Fraction of requirements changed. This measures the density of change requests initiated during the software development lifecycle. Changing a large fraction of the requirements indicate a poor development process that is being driven by unstable requirements. Many change requests may also indicate a poor development process that resulted in errors that are found later in the lifecycle. This later interpretation is not included in the model because there are other, more direct indicators of requirements errors (function point analysis, total defects, cause and effect analysis, and fault number days).
Modeled Range:	0 to 2 change requests per requirements.
Use:	Evidence of the quality of the development process.
Comments:	The more typically used measure is total number of requirement changes. In this the model the density of requirement changes is used to help account for differences between projects of different sizes. The modeled range accounts for the fact that in some processes certain requirements may be changed many times.
Reference:	Moller 1993, Jones 1991



Name:	**CMM Level R**
Type:	Discrete
Observable:	Yes
Phase where Measurable:	All
Phase that it tells about:	Requirements
Meaning:	Software Capability Maturity Model measure for the requirements phase. Measures the predictability, effectiveness, and control of a project software processes. Two states are modeled: 1) CMM ≥ Level 3 — The software development process is defined, documented, and standardized. 2) CMM < Level 3 — The software development process is not defined. Success may still be possible based upon the quality of individual efforts or informal repetition of past successful practices.
Modeled Range:	Probability of state 0 to 1.
Use:	Evidence of the quality of the development process. Evidence of the quality of the V&V process.
Comments:	CMM typically describes the quality of the overall process regardless of phase. The model, however, includes the CMM measure separately for each phase. This allows for the possibility that the process maturity may change (hopefully improve) during the development process. This assumption also simplifies the modeling. The model could easily separately consider each of the five states described by the CMM measure. Currently, no significant benefit is seen from finer modeling.
Reference:	Paulk, Curtis, Chrissis, and Weber 2/1993, Paulk, Weber, Garcia, Chrissis, and Bush 1993, Paulk, Curtis, Chrissis, and Weber 7/1993

Probability Table

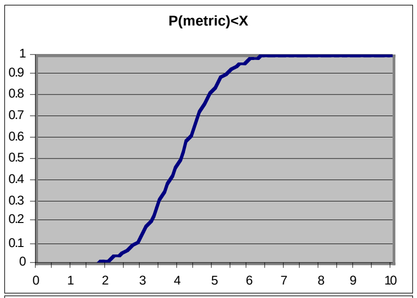
V&V	Good		Poor	
Development Process	Good	Poor	Good	Poor
< Level 3	0.01	0.95	0.95	0.99
≥ Level 3	0.99	0.05	0.05	0.01

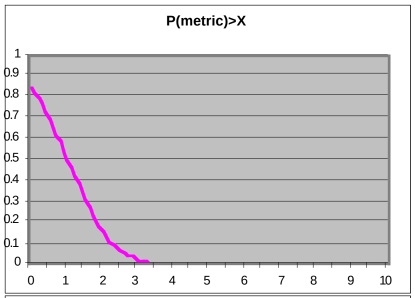
Name:	**V&V R**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the requirements verification and validation process. The node models two states: 1) Good – The requirements V&V process is good enough to detect all safety-significant errors in a reasonably good initial requirements document. The implication is that the V&V process can detect all requirements errors they are not too numerous. It may also be important to consider the skill of the V&V team with respect to the development team. The V&V team must be capable of thoroughly understanding the development team's products. This is not modeled because there are no metrics to give evidence for the V&V team's skill. 2) Poor – The requirements V&V process is not good enough to detect all safety-significant errors in a reasonably good initial requirements document. State 2 is the complement of state 1.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed V&V process.
Use:	Influences the quality of the final requirements.
Comments:	
Reference:	

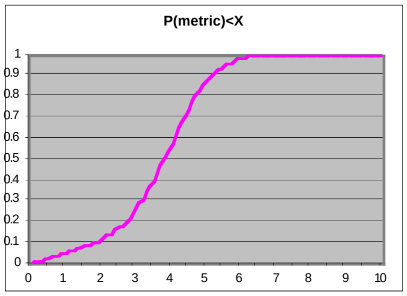
Probability Table

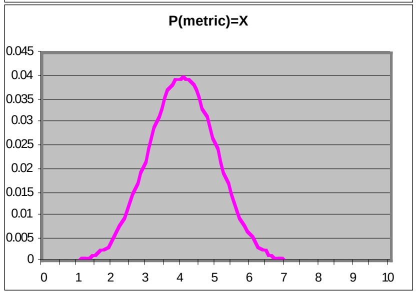
Good	0.9
Poor	0.1

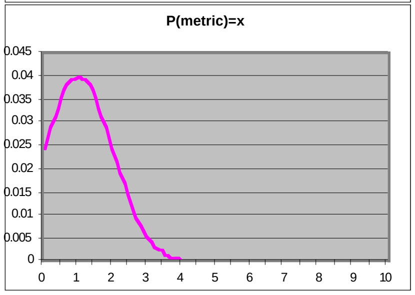
Name:	**mh/Major Defect R**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Man-hours or V&V effort per major defect detected in the requirements. The measure is applied to new code development. A low number (less than 3 hours) indicates potential problems with the requirements. A high number (greater than 5 hours) indicates potential problems with the V&V process.
Modeled Range:	Man-hours between major defect between 0 and 10 hours. A measure greater than 10 indicates either an extremely good initial requirements document or a failed V&V process. If it can be determined that the former is the case, the state of the requirements document may be directly entered into the model, thus bypassing this measure.
Use:	Evidence of the quality of the initial requirements document. Evidence of the quality of the requirements V&V process
Comments:	
Reference:	IEEE 982

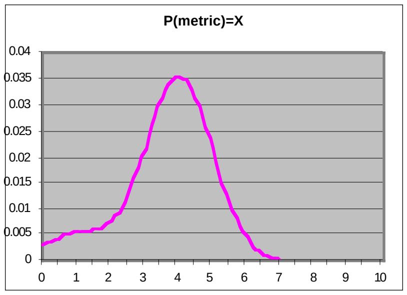










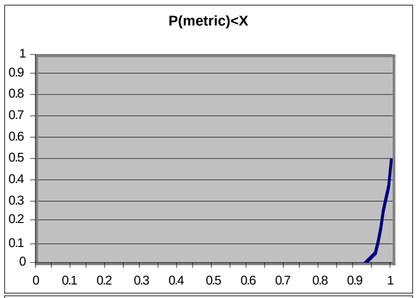


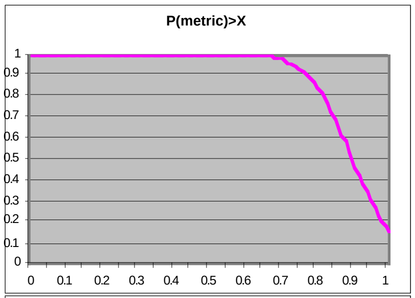
Name:	**Final Requirements**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the final requirements. The node models two states: 1) Good – All safety-significant requirements from the system requirements phase are completely, and correctly addressed in the requirements document. 2) Poor – One or more safety-significant requirements from the system requirements phase is omitted or incorrectly addressed in the requirements document.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the safety of the requirements. Evidence of the quality of the development process. Evidence of the quality of the V&V process. Evidence of the quality of the initial requirements.
Comments:	
Reference:	IEEE 982

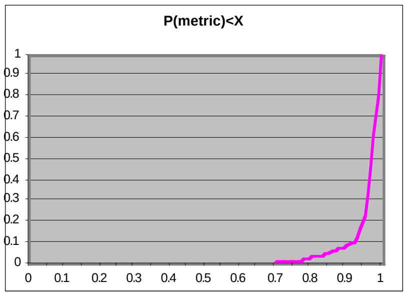
Probability Table

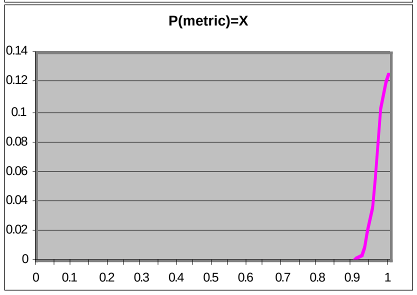
Initial Requirements	Good				Poor			
Development Process	Good		Poor		Good		Poor	
V&V	Good	Poor	Good	Poor	Good	Poor	Good	Poor
Good	0.98	0.89	0.69	0.49	0.89	0.81	0.63	0.45
Poor	0.02	0.11	0.31	0.51	0.11	0.19	0.37	0.55

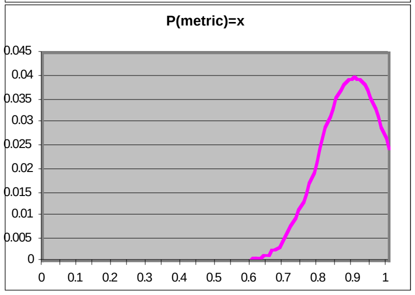
Name:	**Final FPA R**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Function point analysis quality measure on final requirements. Measures 1 minus defect density per function point. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the requirements.
Modeled Range:	FPA 0.9 to 1. An initial FPA < 0.9 probably indicates a failed development process.
Use:	Evidence of the quality of the final requirements.
Comments	
Reference:	IEEE 982

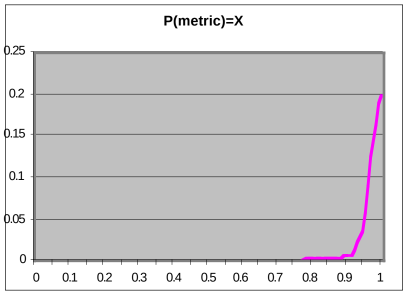






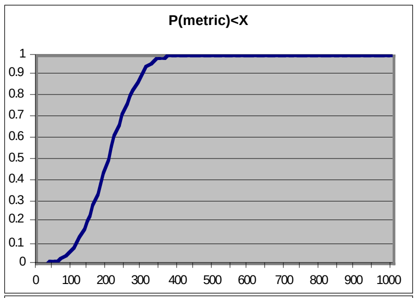


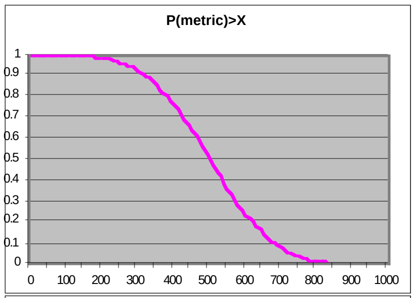


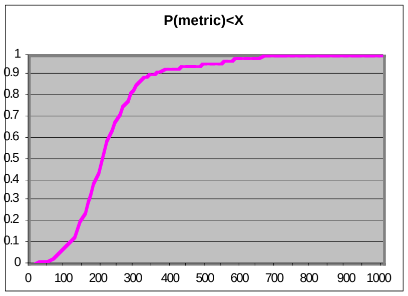


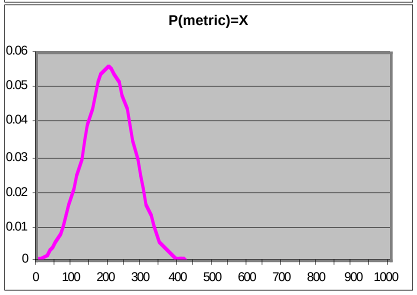
Name:	**Fault Number Phases R**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Architectural Design
	Detailed Design
	Implementation
	Validation
	Operation
Phase that it tells about:	Requirements
Meaning:	The total number of phases that faults remain in the requirements. For this model it is presumed that the measure is after the requirements phase is complete since the number of days it takes to remove a fault within a phase is likely more characteristic of the process structure than the process quality. Consequently, this measure is not germane to initial requirements. If the V&V process is good, this measure indicates the subtlety of errors. If the V&V process is poor, this measure reflects on the quality of the V&V process and the quality of the final product.
Modeled Range:	0 to 1000 fault-phases
Use:	Evidence of Development Process Quality
	Evidence of V&V Process Quality
Comments:	Possibly the distributions associated with this node should be dependent upon the phase in which the measurement is taken.
	The IEEE 982 measure is fault number days. The measure used in this model substitutes phase for days as the duration measure. This will help make the measure less dependent upon project size and staffing.
Reference:	IEEE 982

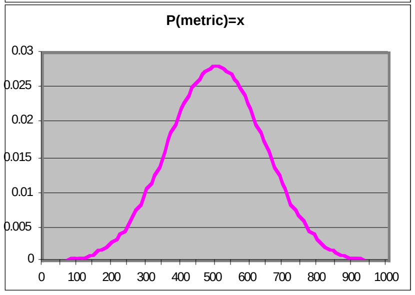
Appendix A

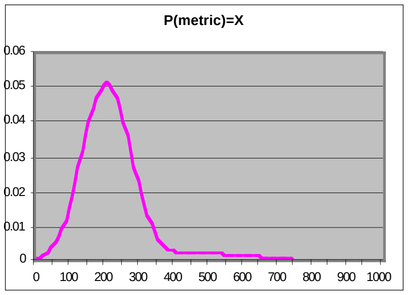

P(metric)<X
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 100 200 300 400 500 600 700 800 900 1000


P(metric)>X
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 100 200 300 400 500 600 700 800 900 1000

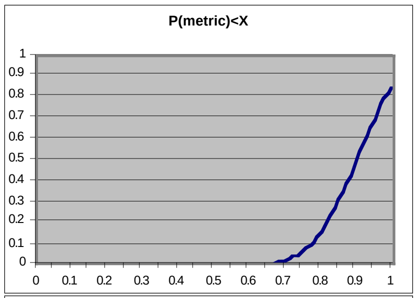

P(metric)<X
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 100 200 300 400 500 600 700 800 900 1000

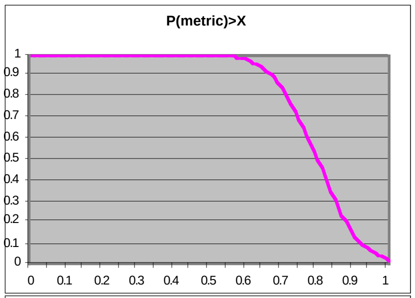

P(metric)=X
0.06
0.05
0.04
0.03
0.02
0.01
0
0 100 200 300 400 500 600 700 800 900 1000

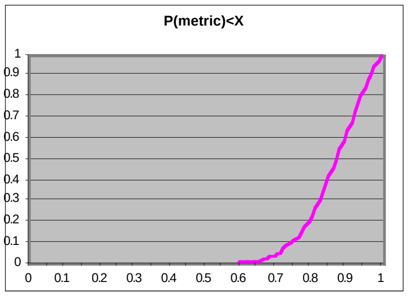

P(metric)=x
0.03
0.025
0.02
0.015
0.01
0.005
0
0 100 200 300 400 500 600 700 800 900 1000

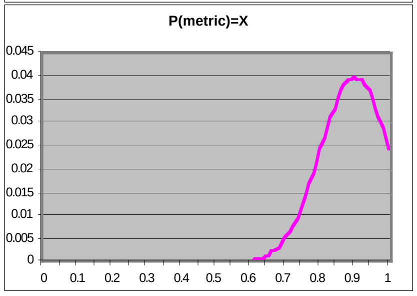

P(metric)=X
0.06
0.05
0.04
0.03
0.02
0.01
0
0 100 200 300 400 500 600 700 800 900 1000

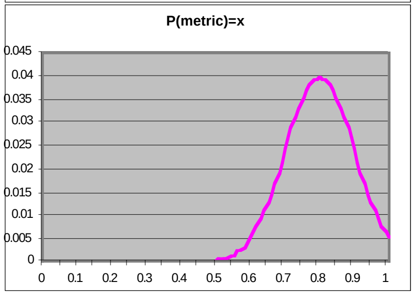
Name:	**Initial Cause & Effect**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Measures the percentage of unambiguous requirements in the initial requirements document. Some ambiguities may result from incomplete graphing of cause and effect relationships, so this is also a measure of completeness.
Modeled Range:	0 to 1
Use:	Evidence of initial requirements quality.
Comments:	
Reference:	IEEE 982

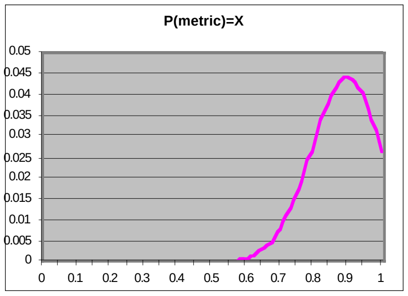




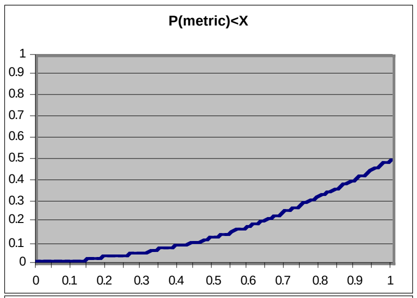


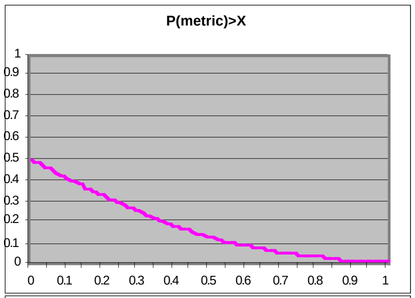


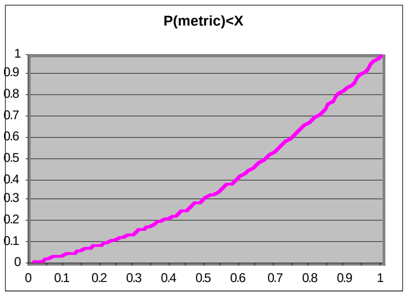


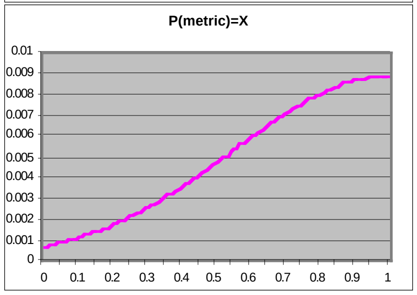


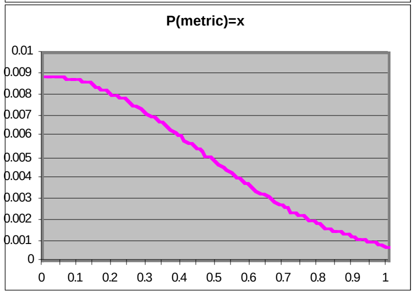
Name:	**Rev., Insp, & Wthroughs R**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Measures the percentage of different types of V&V methods used in the requirements phase.
Modeled Range:	0 to 1
Use:	Evidence of V&V process quality
Comments:	
Reference:	Moller 1993, Freedman 1990, Redmill 1988

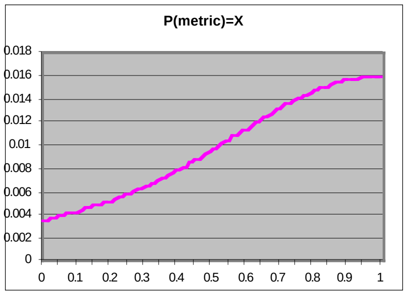




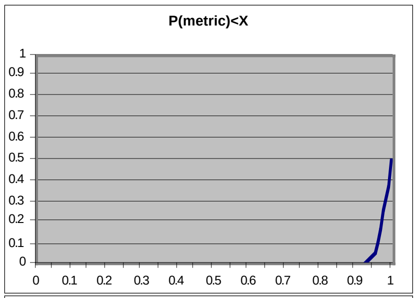


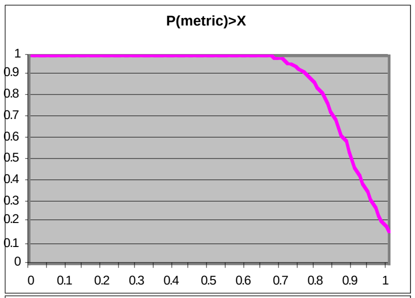


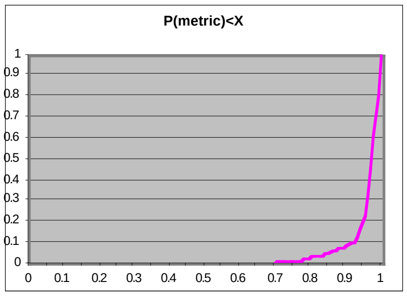


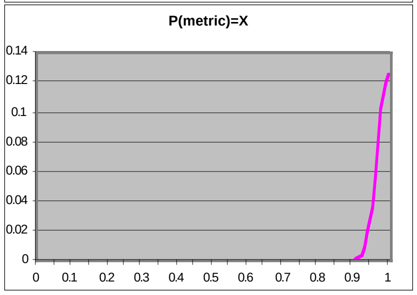


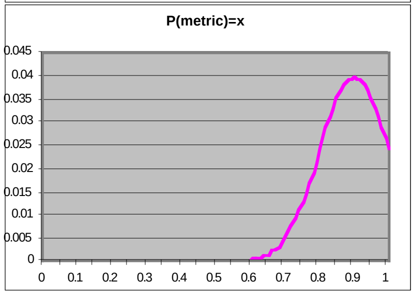
Name:	**Final Cause & Effect**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Measures the percentage of unambiguous requirements in the initial requirements document. Some ambiguities may result from incomplete graphing of cause and effect relationships, so this is also a measure of completeness.
Modeled Range:	0 to 1
Use:	Evidence of initial requirements quality.
Comments:	
Reference:	IEEE 982

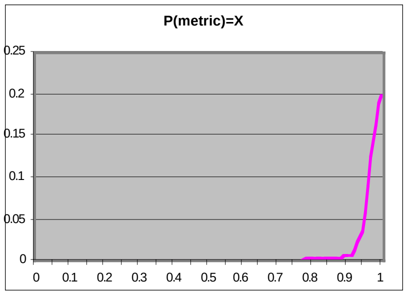






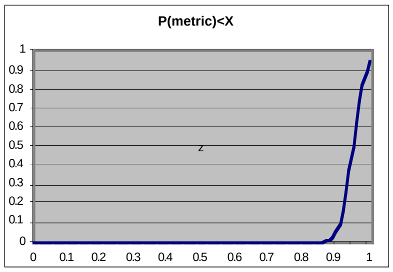


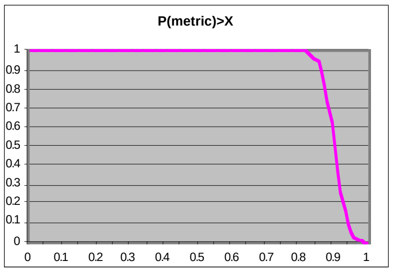


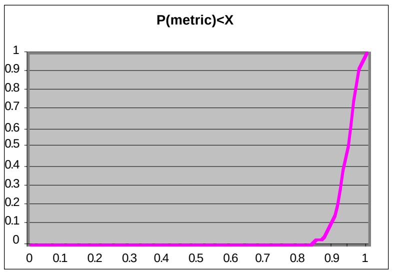


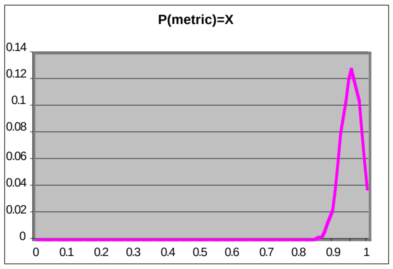
Architectural Design Phase Model

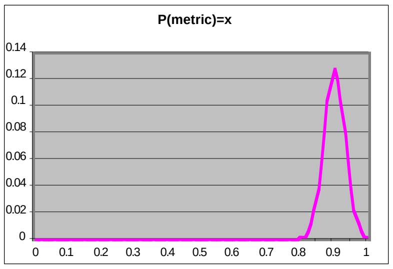
Name:	**Initial FPA A**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Architecture
Phase that it tells about:	Architecture
Meaning:	Function point analysis quality measure on initial draft of architecture. Measures 1 minus defect density per function point. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the architecture.
Modeled Range:	FPA 0.8 to 1. An initial FPA < 0.8 probably indicates a failed development process.
Use:	Evidence of the quality of the initial architecture.
Comments	
Reference:	IEEE 982

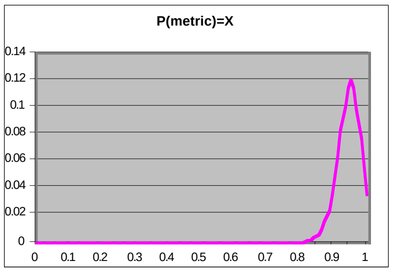












Name:	**Initial Architecture**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the initial architecture. The node models two states: 1) Good – All errors in the initial architecture are detectable by a good V&V process and are correctable by a good development process. The implication is that the errors are not too numerous to be detected and corrected. It may also be important to consider the possibility that errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. 2) Poor – The initial architecture contain some errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1. The probability of the initial architecture being in these states is conditioned by the quality of the development process.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the quality of the final architecture. Evidence of the quality of the development process.
Comments:	
Reference:	IEEE 982

Probability Table

Development Process	Good	Poor
Good	0.99	0.9
Poor	0.01	0.1

Name:	**Development Process A**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Quality of the architecture development process. The node models two states:

1) Good – The architecture development process is good enough that it does not introduce safety-significant errors that cannot be detected by a good V&V process and that it can correct detected errors without introducing new safety-significant errors. The implication is that the development process does not introduce errors that are too numerous to be detected and corrected. It may also be important to consider the possibility that the process introduces errors that are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors.

2) Poor – The architecture development process is expected to produce errors cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1.

The probability of the initial architecture development process being in these states is conditioned by the complexity of the architecture. A more complex design is likely to require a better development process to produce adequate initial and final design.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.

Use: Influences the quality of the initial and final architecture.

Comments:

Reference:

Probability Table

Good	0.9
Poor	0.1

Name:	**CMM Level A**
Type:	Discrete
Observable:	Yes
Phase where Measurable:	All
Phase that it tells about:	Architecture
Meaning:	Software Capability Maturity Model measure for the architectural design phase. Measures the predictability, effectiveness, and control of a project software processes. Two states are modeled: 1) CMM ≥ Level 3 — The software development process is defined, documented, and standardized. 2) CMM < Level 3 — The software development process is not defined. Success may still be possible based upon the quality of individual efforts or informal repetition of past successful practices.
Modeled Range:	Probability of state 0 to 1.
Use:	Evidence of the quality of the development process. Evidence of the quality of the V&V process.
Comments:	CMM typically describes the quality of the overall process regardless of phase. The model, however, includes the CMM measure separately for each phase. This allows for the possibility that the process maturity may change (hopefully improve) during the development process. This assumption also simplifies the modeling. The model could easily separately consider each of the five states described by the CMM measure. Currently, no significant benefit is seen from finer modeling.
Reference:	Paulk, Curtis, Chrissis, and Weber 2/1993, Paulk, Weber, Garcia, Chrissis, and Bush 1993, Paulk, Curtis, Chrissis, and Weber 7/1993

Probability Table

V&V	Good		Poor	
Development Process	Good	Poor	Good	Poor
< Level 3	0.01	0.95	0.95	0.99
≥ Level 3	0.99	0.05	0.05	0.01

Name:	**V&V A**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Quality of the architecture verification and validation process. The node models two states:

1) Good – The architecture V&V process is good enough to detect all safety-significant errors in a reasonably good initial design. The implication is that the V&V process can detect all errors they are not too numerous. It may also be important to consider the skill of the V&V team with respect to the development team. The V&V team must be capable of thoroughly understanding the development team's products. This is not modeled because there are no metrics to give evidence for the V&V team's skill.

2) Poor – The V&V process is not good enough to detect all safety-significant errors in a reasonably good initial architectural design. State 2 is the complement of state 1.

   The probability of the initial architecture V&V process being in these states is conditioned by the complexity of the architecture. A more complex design is likely to require a better V&V to detect errors in the initial.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed V&V process.

Use: Influences the quality of the final architecture.

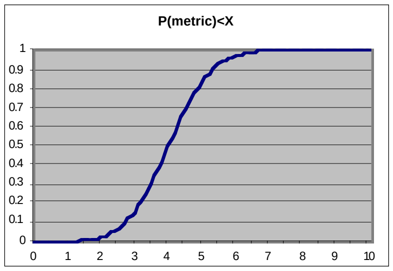
Comments:

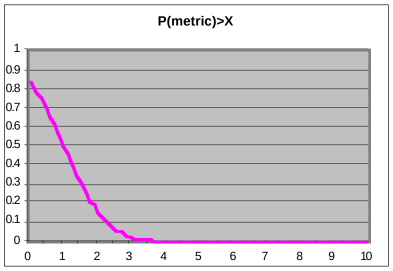
Reference:

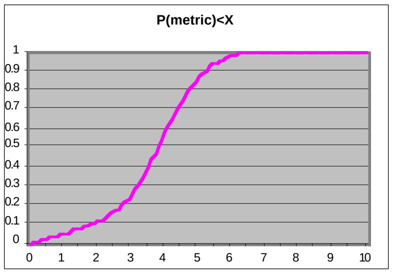
Probability Table

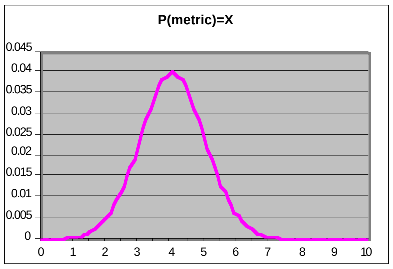
Good	0.9
Poor	0.1

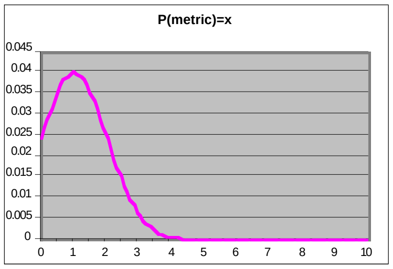
Name:	**mh/Major Defect A**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Architectural Design
Phase that it tells about:	Architectural Design
Meaning:	Man-hours or V&V effort per major defect detected in the architecture. The measure is applied to new code development. A low number (less than 3 hours) indicates potential problems with the requirements. A high number (greater than 5 hours) indicates potential problems with the V&V process.
Modeled Range:	Man-hours between major defect between 0 and 10 hours. A measure greater than 10 indicates either an extremely good initial requirements document or a failed V&V process. If it can be determined that the former is the case, the state of the architecture may be directly entered into the model, thus bypassing this measure.
Use:	Evidence of the quality of the initial architecture.
	Evidence of the quality of the architecture V&V process
Comments:	
Reference:	IEEE 982

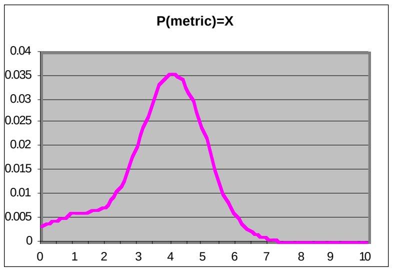










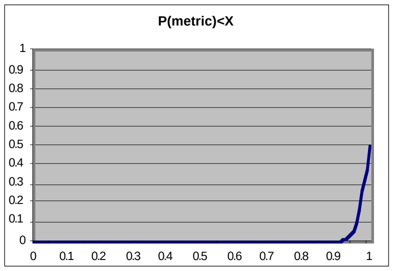
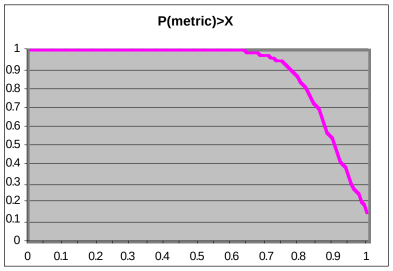
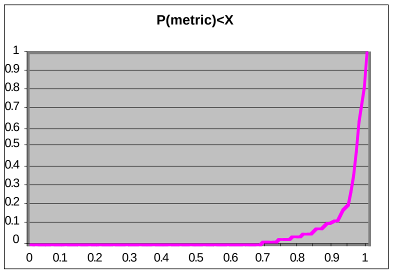
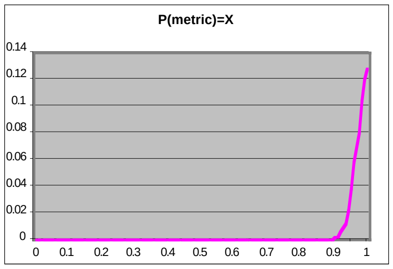
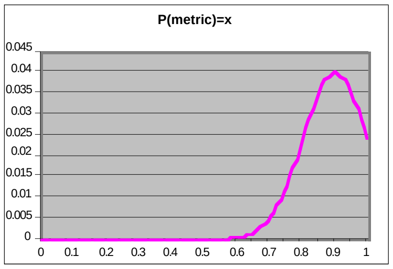
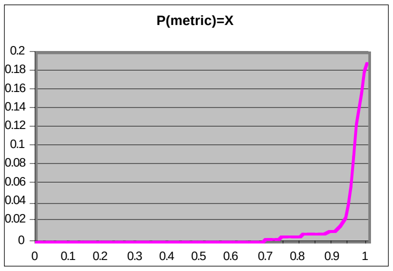


Name:	**Final Architecture**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the final architecture. The node models two states: 1) Good – All safety-significant requirements from the system requirements phase are completely, and correctly addressed in the system architecture. 2) Poor – One or more safety-significant requirements from the system requirements phase is omitted or incorrectly addressed in the system architecture.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the safety of the architecture. Evidence of the quality of the development process. Evidence of the quality of the V&V process. Evidence of the quality of the initial architecture.
Comments:	
Reference:	IEEE 982

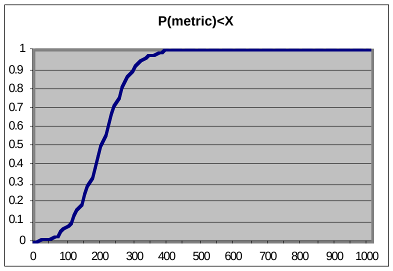
Probability Table

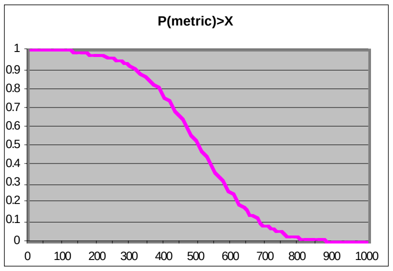
Initial Architecture	Good				Poor			
Development Process	Good		Poor		Good		Poor	
V&V	Good	Poor	Good	Poor	Good	Poor	Good	Poor
Good	0.98	0.89	0.69	0.49	0.89	0.81	0.63	0.45
Poor	0.02	0.11	0.31	0.51	0.11	0.19	0.37	0.55

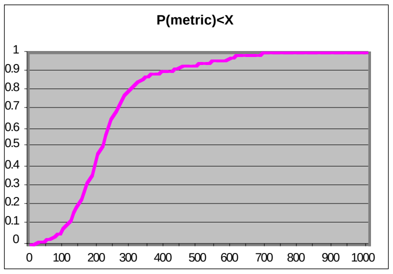
Name:	**Final FPA A**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements
Phase that it tells about:	Requirements
Meaning:	Function point analysis quality measure on final architecture. Measures 1 minus defect density per function point. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	FPA 0.9 to 1. An initial FPA < 0.9 probably indicates a failed development process.
Use:	Evidence of the quality of the final requirements.
Comments	
Reference:	IEEE 982

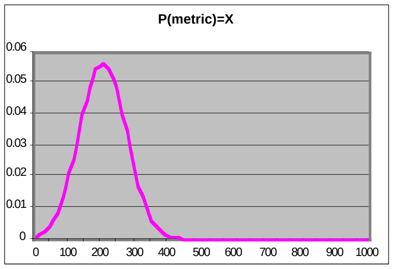
		
		

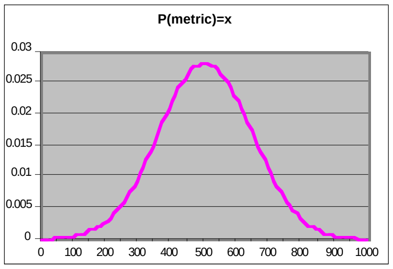
Name:	**Fault Number Phases A**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design Implementation Validation Operation
Phase that it tells about:	Architectural Design
Meaning:	The total number of phases that faults remain in the architecture. For this model it is presumed that the measure is after the architectural design phase is complete since the time required to remove a fault within a phase is likely more characteristic of the process structure than the process quality. Consequently, this measure is not germane to initial requirements. If the V&V process is good, this measure indicates the subtlety of errors. If the V&V process is poor, this measure reflects on the quality of the V&V process and the quality of the final product.
Modeled Range:	0 to 1000 fault-phases
Use:	Evidence of Development Process Quality Evidence of V&V Process Quality
Comments:	Possibly the distributions associated with this node should be dependent upon the phase in which the measurement is taken. The IEEE 982 measure is fault number days. The measure used in this model substitutes phase for days as the duration measure. This will help make the measure less dependent upon project size and staffing.
Reference:	IEEE 982

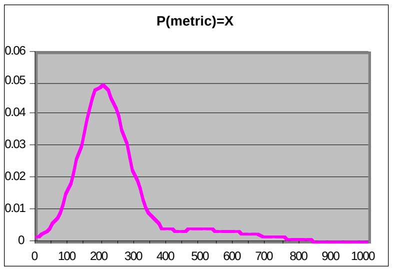




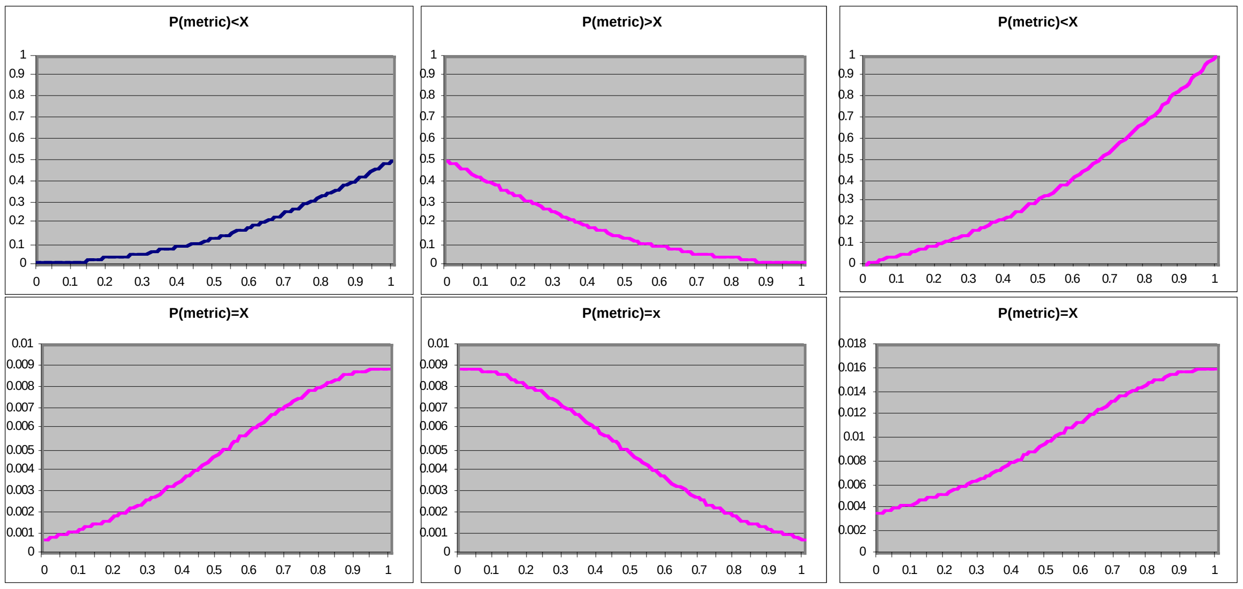




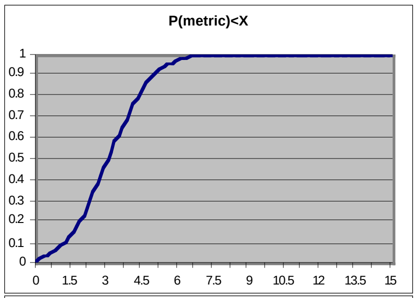


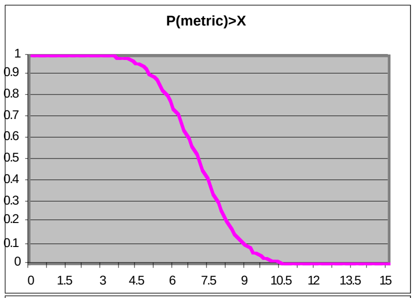


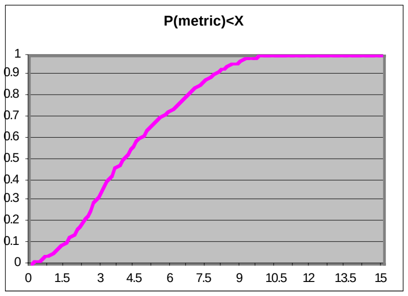
Name:	**Rev., Insp, & Wthroughs A**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Architectural Design
Phase that it tells about:	Architectural Design
Meaning:	Measures the percentage of different types of V&V methods used in the requirements phase.
Modeled Range:	0 to 1
Use:	Evidence of V&V process quality
Comments:	
Reference:	Moller 1993, Freedman 1990, Redmill 1988Name:Name on model
Type:	
Observable:	
Phase where Measurable:	
Phase which it tells about:	
Meaning:	Definition and States
Modeled Range:	
Use:	Used as indicator , Influence on
Reference:	IEEE 982

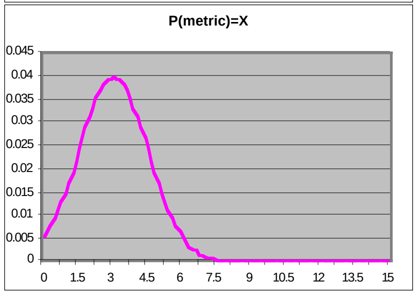


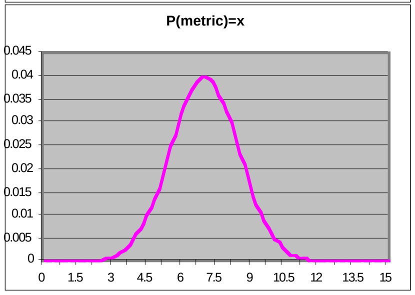
Name:	**Static Archi Complexity**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Architectural Design
Phase that it tells about:	Architectural Design
Meaning:	Measures the complexity of interconnections as the number of interconnections in the software relative to the number of modules.
Modeled Range:	0 to 15. Ten represents the maximum ideal complexity.
Use:	Used as evidence of the actual system complexity in the architectural design phase.
Reference:	IEEE 982

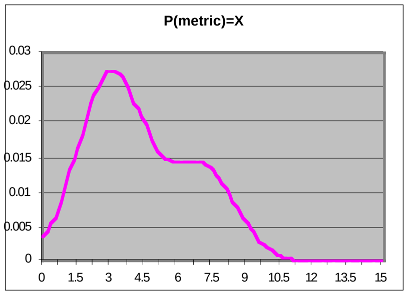




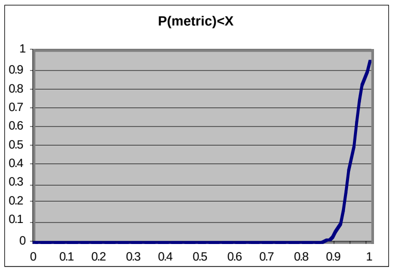


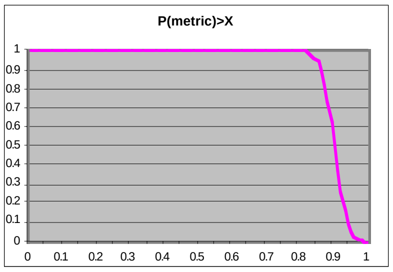


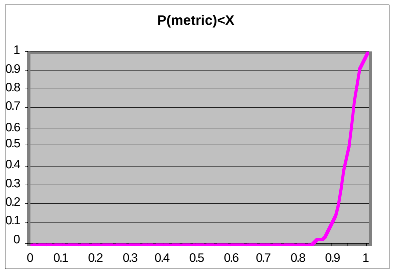


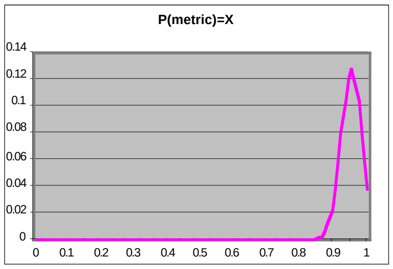


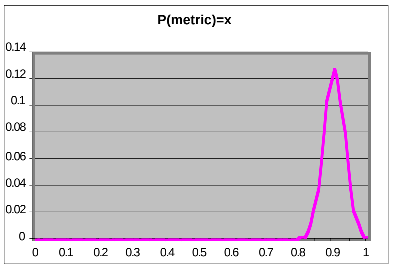
Name:	**Initial Function Point Analysis**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Requirements, Architectural Design
Phase that it tells about:	Requirements, Architectural Design
Meaning:	Describes software product quality in terms of 1 minus the number of defects per software function.
Modeled Range:	0.75 to 1. A value of 1 indicates perfect quality. The measurement for final products should be 1 or very, very close to 1. Values much less than 1 for initial documents indicates a failed development process and further analysis of the software is unwarranted.
Use:	Used in estimation of product quality for initial requirements, final requirements, initial architectural design, and final architectural design.
Reference:	IEEE 982

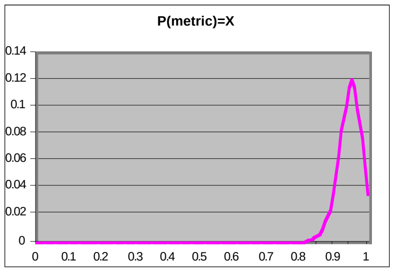




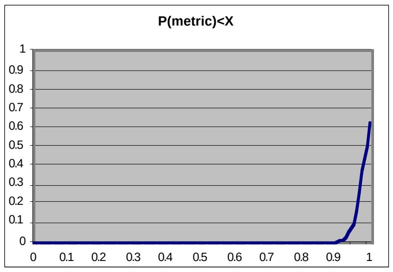


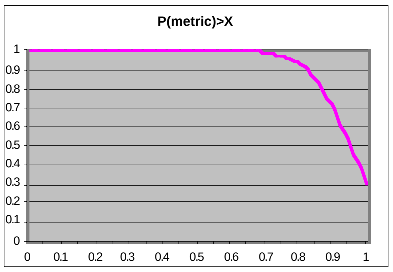


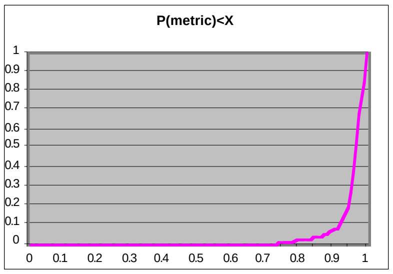


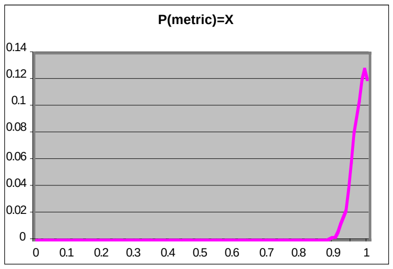


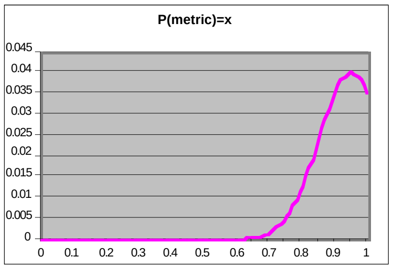
Name:	**Initial Rqmts Traceability**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	Measures the fraction of requirements fulfilled by the initial architecture design.
Modeled Range:	.8 to 1. A value of less than .8 indicates a failed process.
Use:	Evidence of initial architectural design quality.
Reference:	IEEE 982

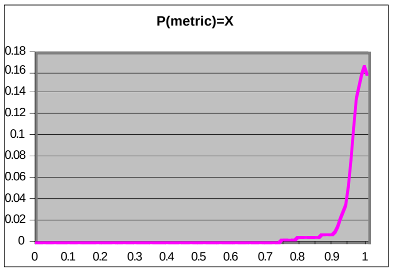






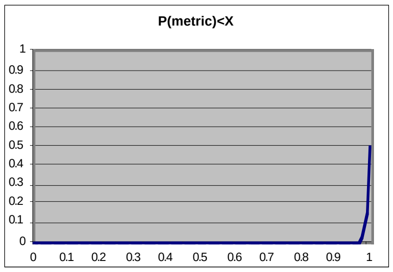


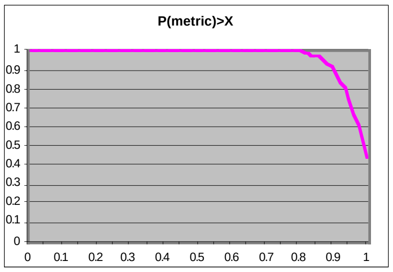


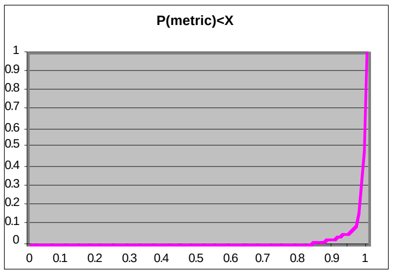


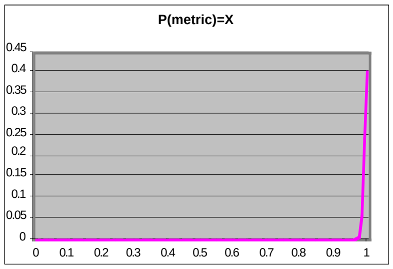
Name:	**Final Rqmts Traceability**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Architecture
Phase that it tells about:	Architecture
Meaning:	Measures the fraction of requirements fulfilled by the final architecture design.
Modeled Range:	.95 to 1. A value of less than .5 indicates a failed process.
Use:	Evidence of initial architectural design quality.
Reference:	IEEE 982

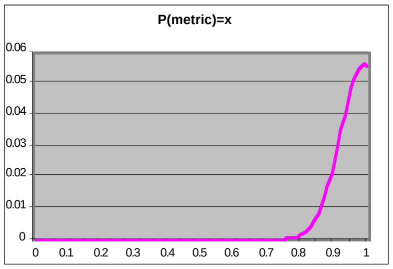
Probability Distribution for Final Requirements Traceability

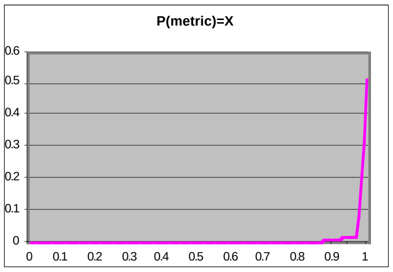












Name:	**Complexity A**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Complexity of the architectural design. The node models three states: 1) Low – The system architecture (e.g., a single stand alone module) should not pose challenges even for a development organization that would rate poorly against BTP-14. 2) Medium – The system architecture (e.g., simple interconnection of a few modules in a single channel) may pose a challenge to a poor development team, but should be within the capability of a development organization that rates well against BTP-14. 3) High – The system architecture (e.g., complex interconnection of redundant channels containing many modules) may be beyond the capability of a development organization that rates well against BTP-14.
Modeled Range:	N/A
Use:	Influences the quality required of the development process. Influences the quality required of the V&V process.
Comments:	
Reference:	IEEE 982

Probability Table

Low	0.33
Medium	0.33
High	0.33

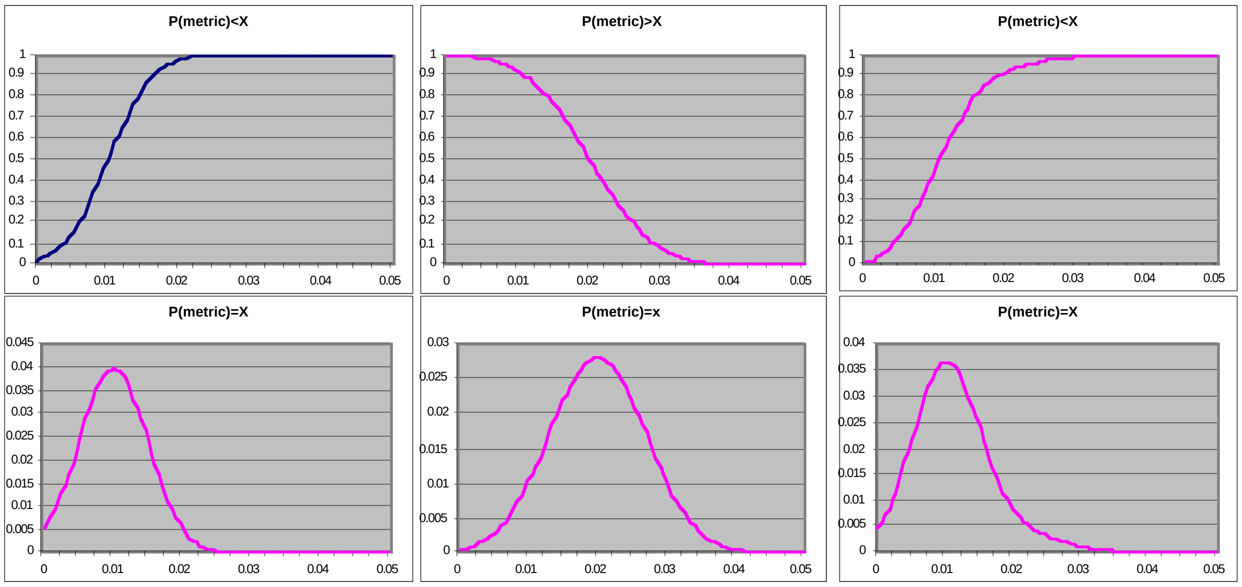
Name:	**Architecture Safety**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Architecture
Meaning:	Quality of the architecture with respect to the fundamental plant safety requirements. The node models two states: 3) Good – All fundamental plant safety requirements are completely, and correctly addressed in the architectural design. 4) Poor – One or more fundamental plant safety requirements omitted or incorrectly addressed in the architectural design.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences safety of the software design.
Comments:	
Reference:	

Probability Table

Requirements Safety	Correct		Incorrect	
Final Architecture	Good	Poor	Good	Poor
Safe	1	0	0	0
Not Safe	0	0	0	0

Detailed Design Phase Model

Name:	**Initial Defect Density D**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	The total number of defects per design statement line. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	0 to 0.05. Initial design densities greater than one per 20 lines probably indicates a failed development process.
Use:	Evidence of the quality of the initial design.
Comments	This measure is often expressed in defects per thousand lines instead of the defects per line expressed here.
Reference:	IEEE 982



Name:	**Initial Design**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the initial design. The node models two states:
	4) Good – All errors in the initial design are detectable by a good V&V process and are correctable by a good development process. The implication is that the errors are not too numerous to be detected and corrected. It may also be important to consider the possibility that errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors.
	5) Poor – The initial architecture contain some errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1.
	The probability of the initial design being in these states is conditioned by the quality of the development process.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the quality of the final architecture.
	Evidence of the quality of the development process.
Comments:	
Reference:	IEEE 982

Probability Table

Development Process	Good	Poor
Good	0.99	0.9
Poor	0.01	0.1

Name:	**Development Process D**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Quality of the detailed design process. The node models two states:

1) Good – The detailed design process is good enough that it does not introduce safety-significant errors that cannot be detected by a good V&V process and that it can correct detected errors without introducing new safety-significant errors. The implication is that the development process does not introduce errors that are too numerous to be detected and corrected. It may also be important to consider the possibility that the process introduces errors that are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors.

2) Poor – The detailed design process is expected to produce errors cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1.

The probability of the detailed design process being in these states is conditioned by the complexity of the design. A more complex design is likely to require a better development process to produce adequate initial and final design.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.

Use: Influences the quality of the initial and final architecture.

Comments:

Reference:

Probability Table

Good	0.9
Poor	0.1

Name:	**CMM Level D**
Type:	Discrete
Observable:	Yes
Phase where Measurable:	All
Phase that it tells about:	Detailed Design
Meaning:	Software Capability Maturity Model measure for the architectural design phase. Measures the predictability, effectiveness, and control of a project software processes. Two states are modeled: 1) CMM ≥ Level 3 — The software development process is defined, documented, and standardized. 2) CMM < Level 3 — The software development process is not defined. Success may still be possible based upon the quality of individual efforts or informal repetition of past successful practices.
Modeled Range:	Probability of state 0 to 1.
Use:	Evidence of the quality of the development process. Evidence of the quality of the V&V process.
Comments:	CMM typically describes the quality of the overall process regardless of phase. The model, however, includes the CMM measure separately for each phase. This allows for the possibility that the process maturity may change (hopefully improve) during the development process. This assumption also simplifies the modeling. The model could easily separately consider each of the five states described by the CMM measure. Currently, no significant benefit is seen from finer modeling.
Reference:	Paulk, Curtis, Chrissis, and Weber 2/1993, Paulk, Weber, Garcia, Chrissis, and Bush 1993, Paulk, Curtis, Chrissis, and Weber 7/1993

Probability Table

V&V	Good		Poor	
Development Process	Good	Poor	Good	Poor
< Level 3	0.01	0.95	0.95	0.99
≥ Level 3	0.99	0.05	0.05	0.01

Name:	**V&V D**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning:

Quality of the detailed design verification and validation process. The node models two states:

1) Good – The detailed design V&V process is good enough to detect all safety-significant errors in a reasonably good initial design. The implication is that the V&V process can detect all errors they are not too numerous. It may also be important to consider the skill of the V&V team with respect to the development team. The V&V team must be capable of thoroughly understanding the development team's products. This is not modeled because there are no metrics to give evidence for the V&V team's skill.

2) Poor – The V&V process is not good enough to detect all safety-significant errors in a reasonably good initial detailed design. State 2 is the complement of state 1.

   The probability of the detailed design V&V process being in these states is conditioned by the complexity of the design. A more complex design is likely to require a better V&V to detect errors in the initial.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed V&V process.

Use: Influences the quality of the final architecture.

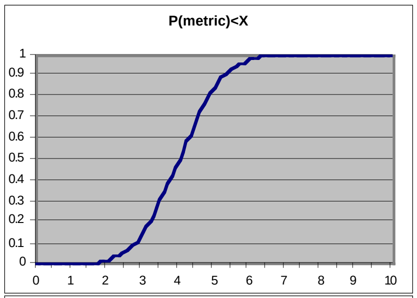
Comments:

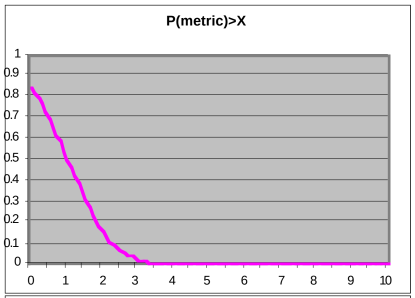
Reference:

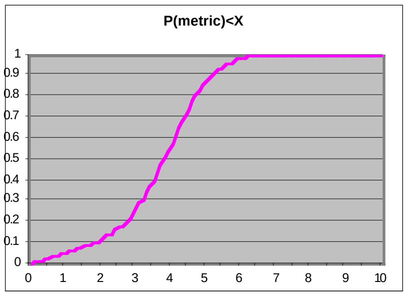
Probability Table

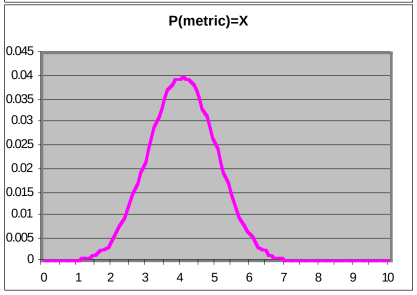
Good	0.9
Poor	0.1

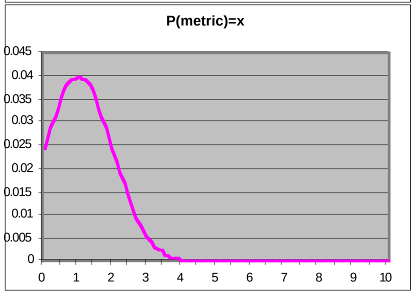
Name:	**mh/Major Defect D**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	Man-hours or V&V effort per major defect detected in the detailed design. The measure is applied to new code development. A low number (less than 3 hours) indicates potential problems with the requirements. A high number (greater than 5 hours) indicates potential problems with the V&V process.
Modeled Range:	Man-hours between major defect between 0 and 10 hours. A measure greater than 10 indicates either an extremely good initial design document or a failed V&V process. If it can be determined that the former is the case, the state of the architecture may be directly entered into the model, thus bypassing this measure.
Use:	Evidence of the quality of the initial detailed design. Evidence of the quality of the detailed design V&V process
Comments:	
Reference:	IEEE 982

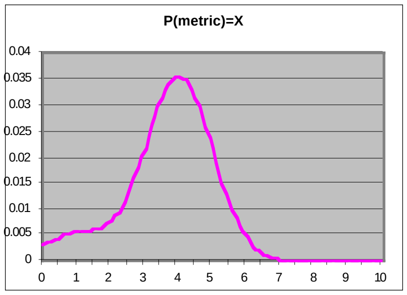










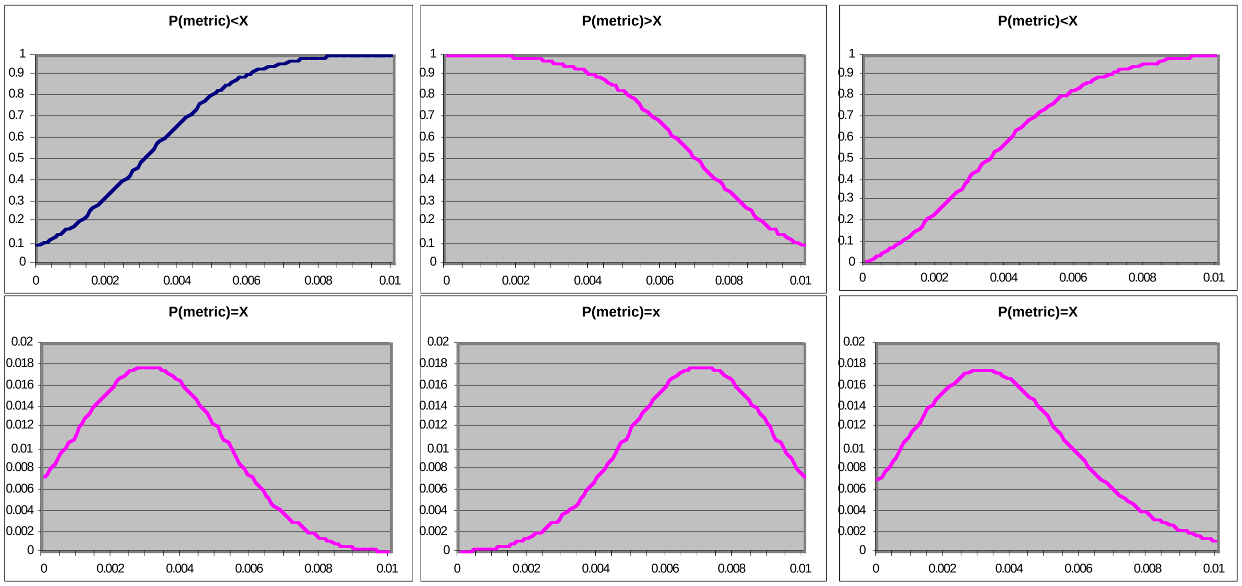


Name:	**Final Design**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the final detailed design. The node models two states: 1) Good – All safety-significant requirements are completely, and correctly addressed in the detailed design. 2) Poor – One or more safety-significant requirements are omitted or incorrectly addressed in the detailed design.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the safety of the detailed design. Evidence of the quality of the development process. Evidence of the quality of the V&V process. Evidence of the quality of the initial detailed design.
Comments:	
Reference:	IEEE 982

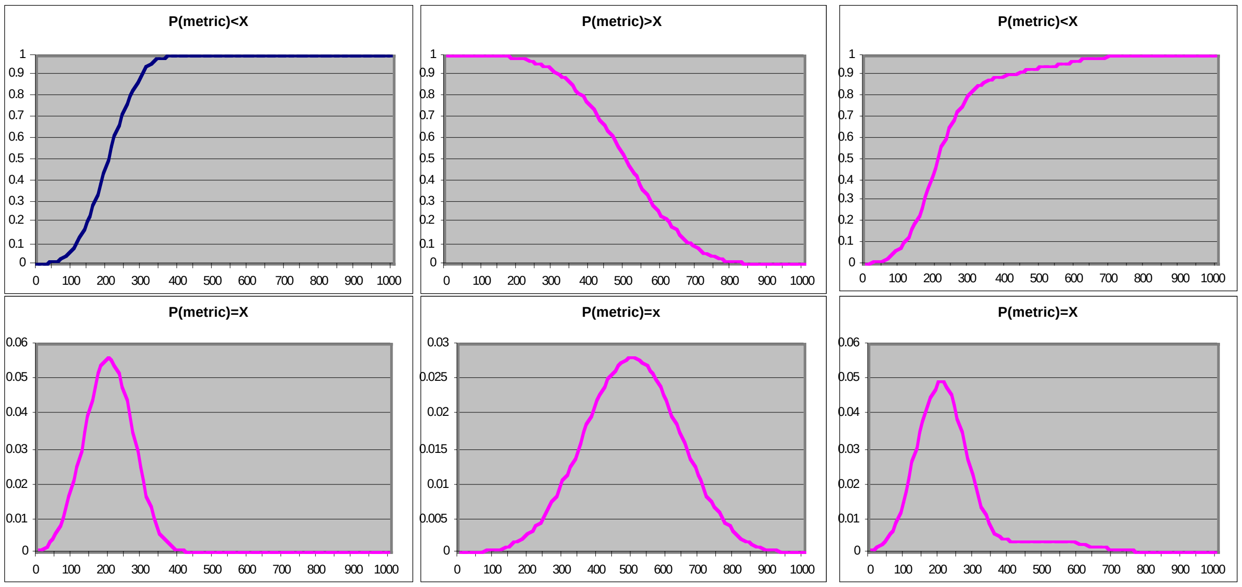
Probability Table

Initial Design	Good				Poor			
Development Process	Good		Poor		Good		Poor	
V&V	Good	Poor	Good	Poor	Good	Poor	Good	Poor
Good	0.98	0.89	0.69	0.49	0.89	0.81	0.63	0.45
Poor	0.02	0.11	0.31	0.51	0.11	0.19	0.37	0.55

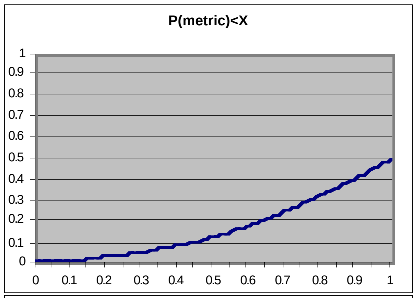
Name:	**Final Defect Density D**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	The total number of defects per design statement line. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	0 to 0.01. Final design densities greater than one per 100 lines probably indicates a failed development process.
Use:	Evidence of the quality of the final design.
Comments	This measure is often expressed in defects per thousand lines instead of the defects per line expressed here.
Reference:	IEEE 982

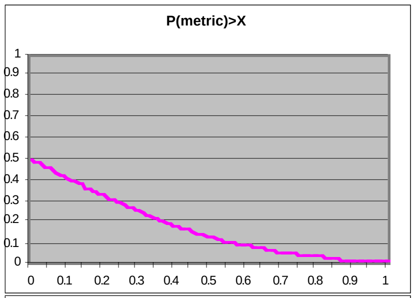


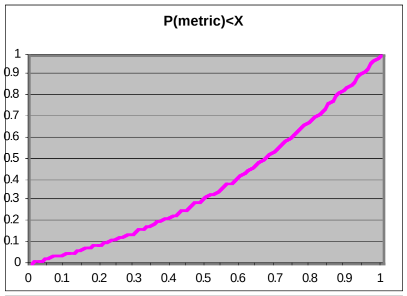
Name:	**Fault Number Phases D**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation Validation Operation
Phase that it tells about:	Detailed Design
Meaning:	The total number of phases that faults remain in the detailed design. For this model it is presumed that the measure is after the detailed design phase is complete since the time required to remove a fault within a phase is likely more characteristic of the process structure than the process quality. Consequently, this measure is not germane to initial requirements. If the V&V process is good, this measure indicates the subtlety of errors. If the V&V process is poor, this measure reflects on the quality of the V&V process and the quality of the final product.
Modeled Range:	0 to 1000 fault-phases
Use:	Evidence of Development Process Quality Evidence of V&V Process Quality
Comments:	Possibly the distributions associated with this node should be dependent upon the phase in which the measurement is taken. The IEEE 982 measure is fault number days. The measure used in this model substitutes phase for days as the duration measure. This will help make the measure less dependent upon project size and staffing.
Reference:	IEEE 982

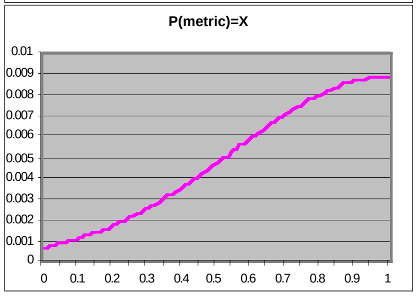


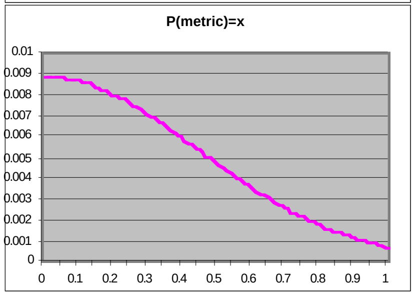
Name:	**Rev., Insp, & Wthroughs D**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	Measures the percentage of different types of V&V methods used in the detailed design phase.
Modeled Range:	0 to 1
Use:	Evidence of V&V process quality
Comments:	
Reference:	Moller 1993, Freedman 1990, Redmill 1988Name:Name on model
Type:	
Observable:	
Phase where Measurable:	
Phase which it tells about:	
Meaning:	Definition and States
Modeled Range:	
Use:	Used as indicator , Influence on
Reference:	IEEE 982

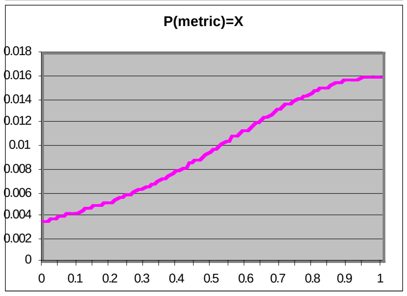




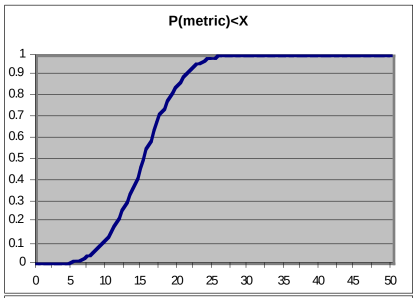


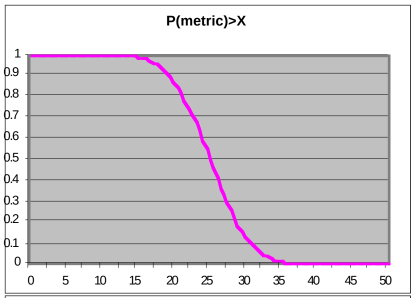


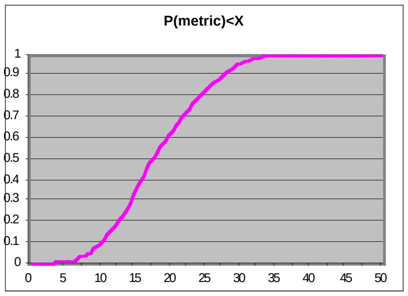


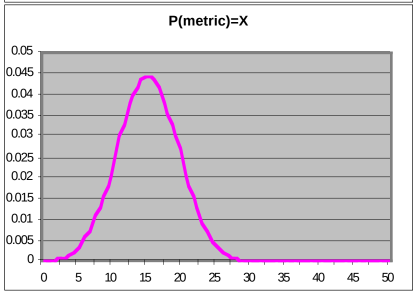


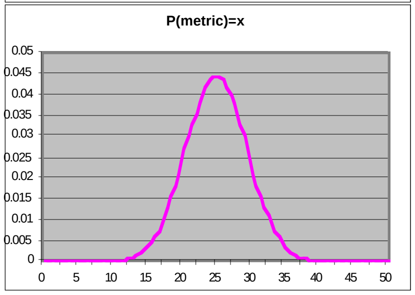
Name:	**System Design Complexity**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	Design complexity as a function of the total number of interconnections and the number of input or output variables per module.
Modeled Range:	1 to 50. 25 represents the average complexity equivalent to the Rome Laboratory fault density measurement for process control systems.
Use:	Used as evidence of the actual system complexity in the detailed design phase.
Reference:	IEEE 982

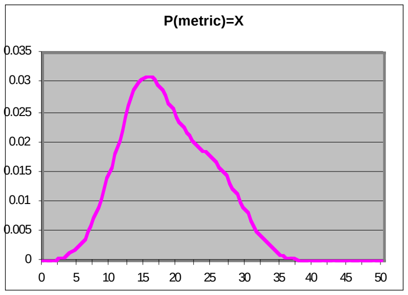




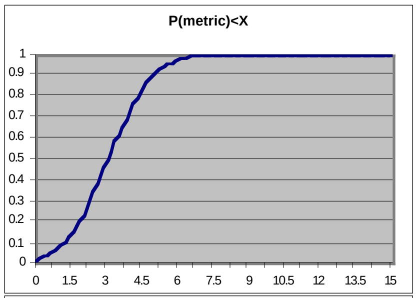


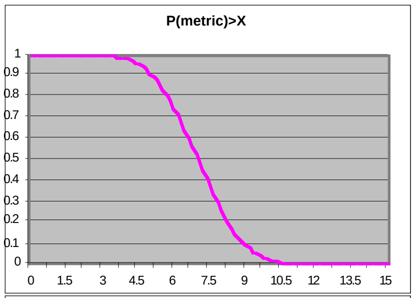


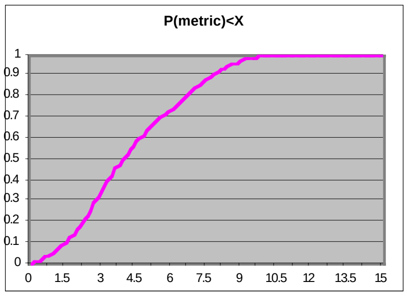


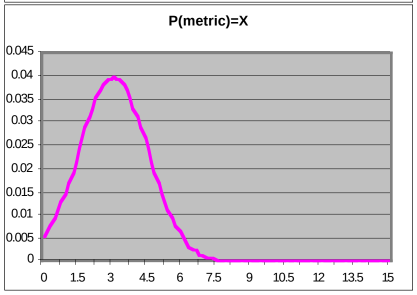


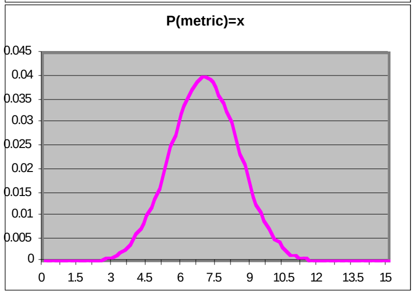
Name:	**Cyclomatic Complexity**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Detailed Design
Meaning:	Describes module complexity in terms of the number of nodes within a module that can transfer control to more than one node. For this model this measure is taken for the most complex module.
Modeled Range:	0 to 15. Ten represents the maximum ideal complexity
Use:	Used as evidence of the actual system complexity.
Reference:	IEEE 982

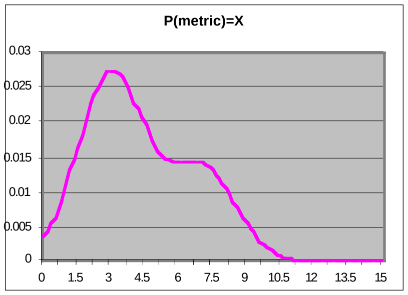












Name:	**Complexity D**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Complexity of the architectural design. The node models three states:

1) Low – The system detailed design should not pose challenges even for a development organization that would rate poorly against BTP-14.

2) Medium – The system detailed design may pose a challenge to a poor development team, but should be within the capability of a development organization that rates well against BTP-14.

3) High – The system detailed design may be beyond the capability of a development organization that rates well against BTP-14.

Modeled Range: N/A

Use: Influences the quality required of the development process.

Influences the quality required of the V&V process.

Comments:

Reference:

Probability Table

Low	0.33
Medium	0.33
High	0.33

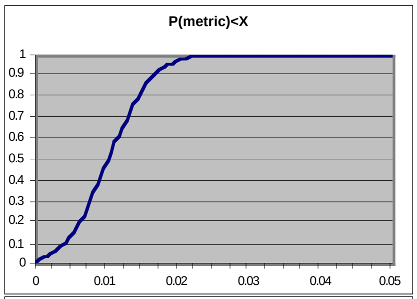
Name:	**Design Safety**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Design
Meaning:	Quality of the detailed design with respect to the fundamental plant safety requirements. The node models two states: 1) Good – All fundamental plant safety requirements are completely, and correctly addressed in the detailed design. 2) Poor – One or more fundamental plant safety requirements omitted or incorrectly addressed in the detailed design.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences safety of the software implementation.
Comments:	
Reference:	

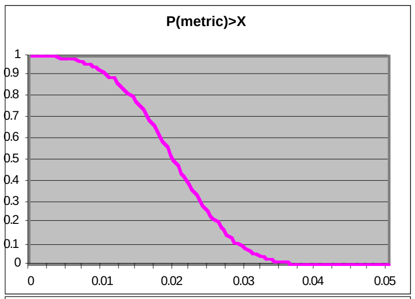
Probability Table

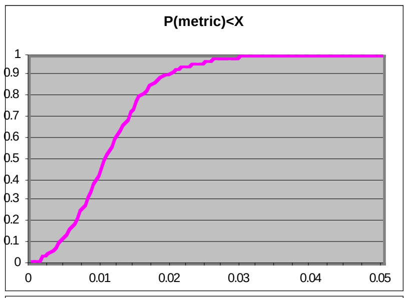
Architecture Safety	Correct		Incorrect	
Final Design	Good	Poor	Good	Poor
Safe	1	0	0	0
Not Safe	0	0	0	0

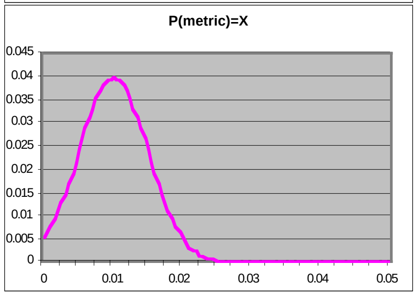
Implementation Phase Model

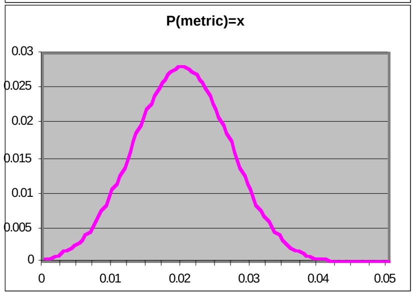
Name:	**Initial Defect Density C**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation
Phase that it tells about:	Implementation
Meaning:	The total number of defects per line of code. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	0 to 0.05. Initial design densities greater than one per 20 lines probably indicates a failed development process.
Use:	Evidence of the quality of the initial code.
Comments	This measure is often expressed in defects per thousand lines instead of the defects per line expressed here.
Reference:	IEEE 982

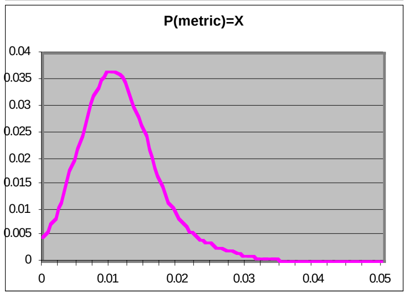












Name:	**Initial Code**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the initial code. The node models two states: 1) Good – All errors in the initial code are detectable by a good V&V process and are correctable by a good development process. The implication is that the errors are not too numerous to be detected and corrected. It may also be important to consider the possibility that errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. 2) Poor – The initial corrected contain some errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1. The probability of the initial design being in these states is conditioned by the quality of the development process.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the quality of the final code. Evidence of the quality of the development process.
Comments:	
Reference:	IEEE 982

Probability Table

Development Process	Good	Poor
Good	0.99	0.9
Poor	0.01	0.1

Name:	**Development Process C**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning:

Quality of the code development process. The node models two states:

1) Good – The code development process is good enough that it does not introduce safety-significant errors that cannot be detected by a good V&V process and that it can correct detected errors without introducing new safety-significant errors. The implication is that the development process does not introduce errors that are too numerous to be detected and corrected. It may also be important to consider the possibility that the process introduces errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors.

2) Poor – The code development process is expected to produce errors cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1.

The probability of the code development process being in these states is conditioned by the complexity of the code. More complex code is likely to require a better development process to produce adequate initial and final design.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.

Use: Influences the quality of the initial and final code.

Comments:

Reference:

Probability Table

Good	0.9
Poor	0.1

Name:	**CMM Level C**
Type:	Discrete
Observable:	Yes
Phase where Measurable:	All
Phase that it tells about:	Implementation
Meaning:	Software Capability Maturity Model measure for the implementation phase. Measures the predictability, effectiveness, and control of a project software processes. Two states are modeled: 1) CMM ≥ Level 3 — The software development process is defined, documented, and standardized. 2) CMM < Level 3 — The software development process is not defined. Success may still be possible based upon the quality of individual efforts or informal repetition of past successful practices.
Modeled Range:	Probability of state 0 to 1.
Use:	Evidence of the quality of the development process. Evidence of the quality of the V&V process.
Comments:	CMM typically describes the quality of the overall process regardless of phase. The model, however, includes the CMM measure separately for each phase. This allows for the possibility that the process maturity may change (hopefully improve) during the development process. This assumption also simplifies the modeling. The model could easily separately consider each of the five states described by the CMM measure. Currently, no significant benefit is seen from finer modeling.
Reference:	Paulk, Curtis, Chrissis, and Weber 2/1993, Paulk, Weber, Garcia, Chrissis, and Bush 1993, Paulk, Curtis, Chrissis, and Weber 7/1993

Probability Table

V&V	Good		Poor	
Development Process	Good	Poor	Good	Poor
< Level 3	0.01	0.95	0.95	0.99
≥ Level 3	0.99	0.05	0.05	0.01

Name:	**V&V C**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Quality of the code verification and validation process. The node models two states:

1) Good – The code V&V process is good enough to detect all safety-significant errors in reasonably good initial code. The implication is that the V&V process can detect all errors they are not too numerous. It may also be important to consider the skill of the V&V team with respect to the development team. The V&V team must be capable of thoroughly understanding the development team's products. This is not modeled because there are no metrics to give evidence for the V&V team's skill.

2) Poor – The V&V process is not good enough to detect all safety-significant errors in reasonably good initial code. State 2 is the complement of state 1.

   The probability of the detailed design V&V process being in these states is conditioned by the complexity of the code. More complex code is likely to require a better V&V to detect errors in the initial.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed V&V process.

Use: Influences the quality of the final code.

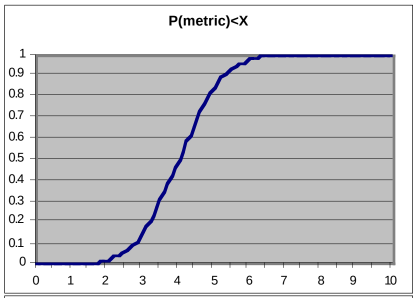
Comments:

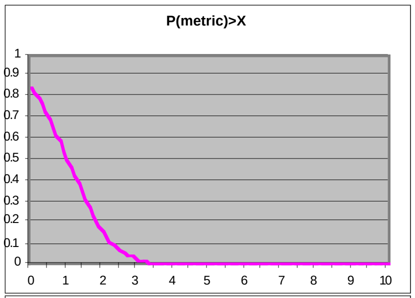
Reference:

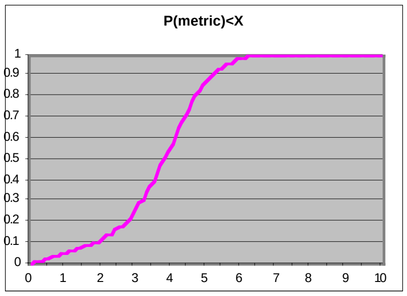
Probability Table

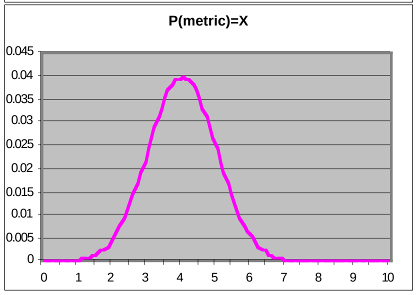
Good	0.9
Poor	0.1

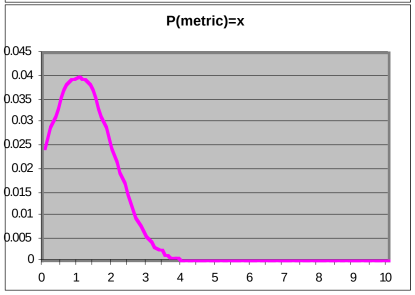
Name:	**mh/Major Defect C**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation
Phase that it tells about:	Implementation
Meaning:	Man-hours or V&V effort per major defect detected in the code. The measure is applied to new code development. A low number (less than 3 hours) indicates potential problems with the requirements. A high number (greater than 5 hours) indicates potential problems with the V&V process.
Modeled Range:	Man-hours between major defect between 0 and 10 hours. A measure greater than 10 indicates either an extremely good initial design document or a failed V&V process. If it can be determined that the former is the case, the state of the architecture may be directly entered into the model, thus bypassing this measure.
Use:	Evidence of the quality of the initial code.
	Evidence of the quality of the code V&V process
Comments:	
Reference:	IEEE 982

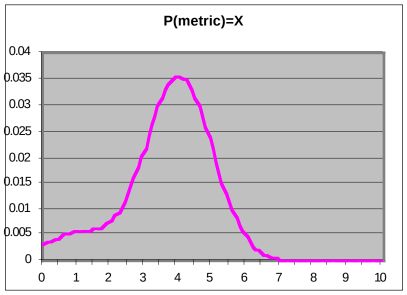










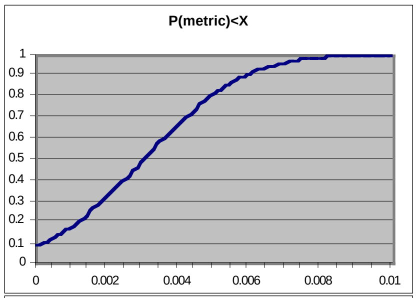


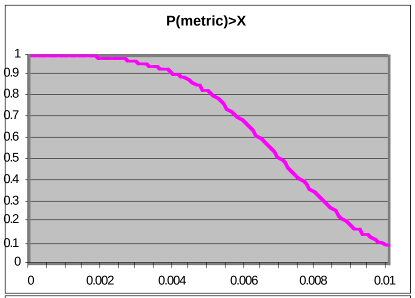
Name:	**Final Code**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the final code. The node models two states: 1) Good – All safety-significant requirements are completely, and correctly addressed in the code. 2) Poor – One or more safety-significant requirements are omitted or incorrectly addressed in the code.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the safety of the code. Evidence of the quality of the development process. Evidence of the quality of the V&V process. Evidence of the quality of the initial code.
Comments:	
Reference:	IEEE 982

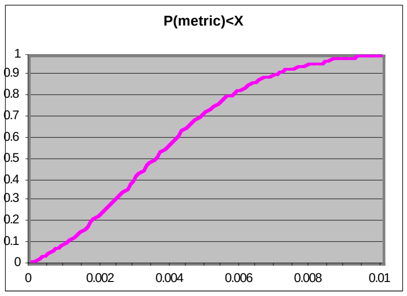
Probability Table

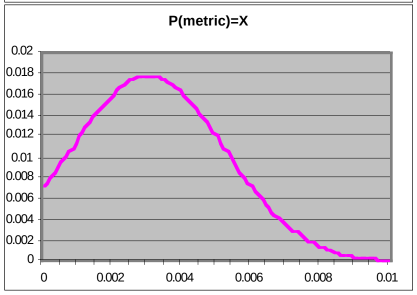
Initial Architecture	Good				Poor			
Development Process	Good		Poor		Good		Poor	
V&V	Good	Poor	Good	Poor	Good	Poor	Good	Poor
Good	0.98	0.89	0.69	0.49	0.89	0.81	0.63	0.45
Poor	0.02	0.11	0.31	0.51	0.11	0.19	0.37	0.55

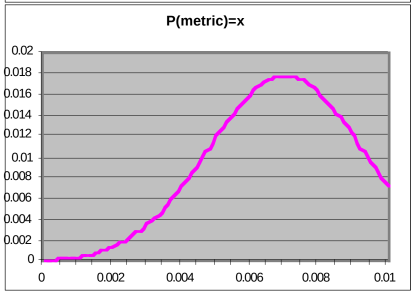
Name:	**Final Defect Density C**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation
Phase that it tells about:	Implementation
Meaning:	The total number of defects per code statement line. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	0 to 0.01. Final defect densities greater than one per 100 lines probably indicates a failed development process.
Use:	Evidence of the quality of the final code.
Comments	This measure is often expressed in defects per thousand lines instead of the defects per line expressed here.
Reference:	IEEE 982

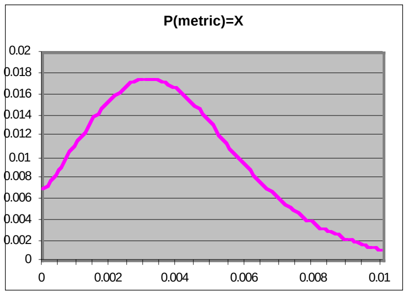




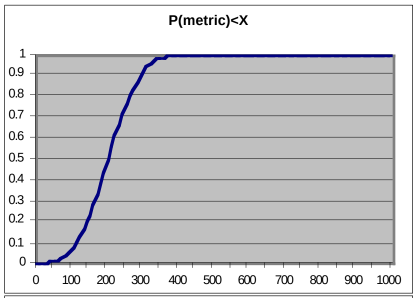


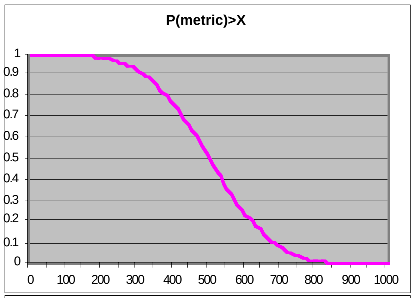


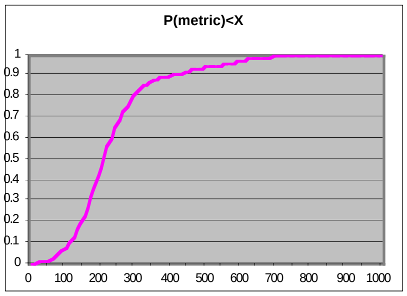


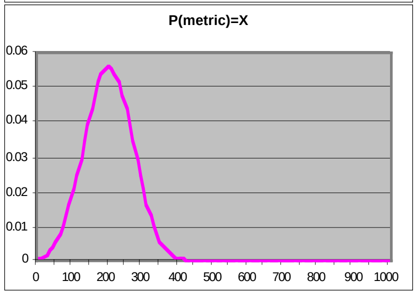


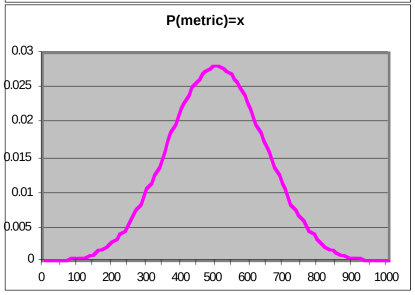
Name:	**Fault Number Phases C**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Validation
	Operation
Phase that it tells about:	Code
Meaning:	The total number of phases that faults remain in the code. For this model it is presumed that the measure is after the implementation phase is complete since the time required to remove a fault within a phase is likely more characteristic of the process structure than the process quality. Consequently, this measure is not germane to initial requirements. If the V&V process is good, this measure indicates the subtlety of errors. If the V&V process is poor, this measure reflects on the quality of the V&V process and the quality of the final product.
Modeled Range:	0 to 1000 fault-phases
Use:	Evidence of Development Process Quality
	Evidence of V&V Process Quality
Comments:	Possibly the distributions associated with this node should be dependent upon the phase in which the measurement is taken.
	The IEEE 982 measure is fault number days. The measure used in this model substitutes phase for days as the duration measure. This will help make the measure less dependent upon project size and staffing.
Reference:	IEEE 982

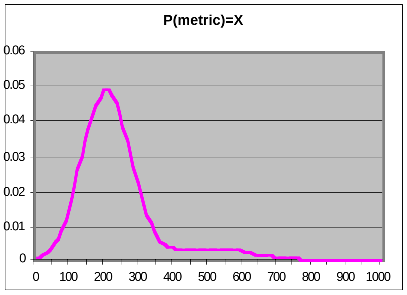

P(metric)<X
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
0 100 200 300 400 500 600 700 800 900 1000


P(metric)>X
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
0 100 200 300 400 500 600 700 800 900 1000

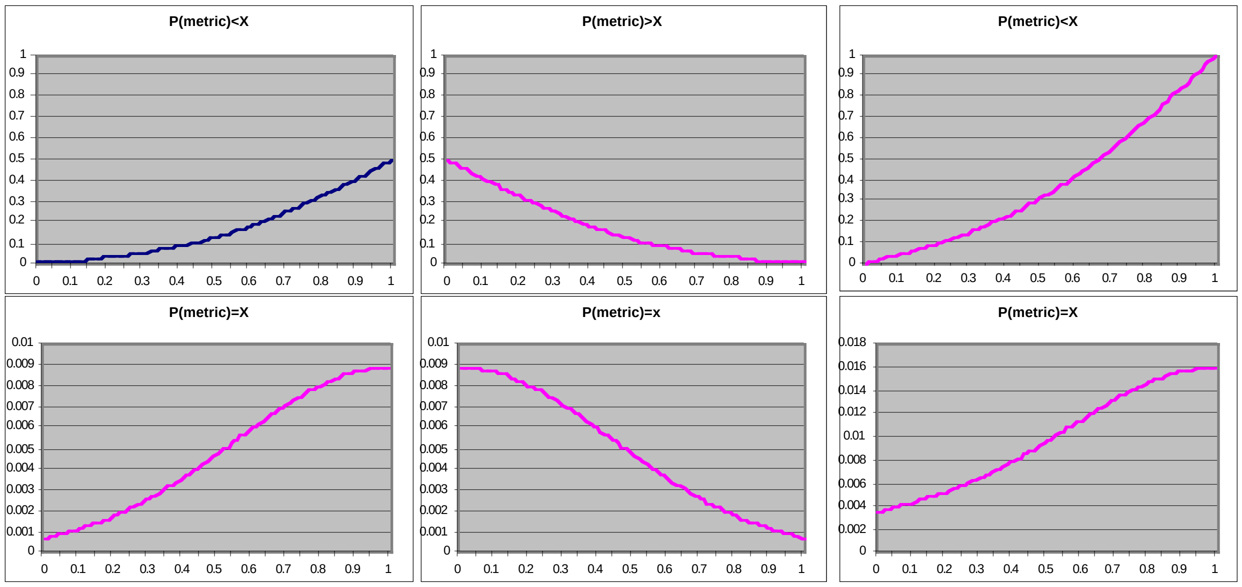

P(metric)<X
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
0 100 200 300 400 500 600 700 800 900 1000


P(metric)=X
0 0.01 0.02 0.03 0.04 0.05 0.06
0 100 200 300 400 500 600 700 800 900 1000

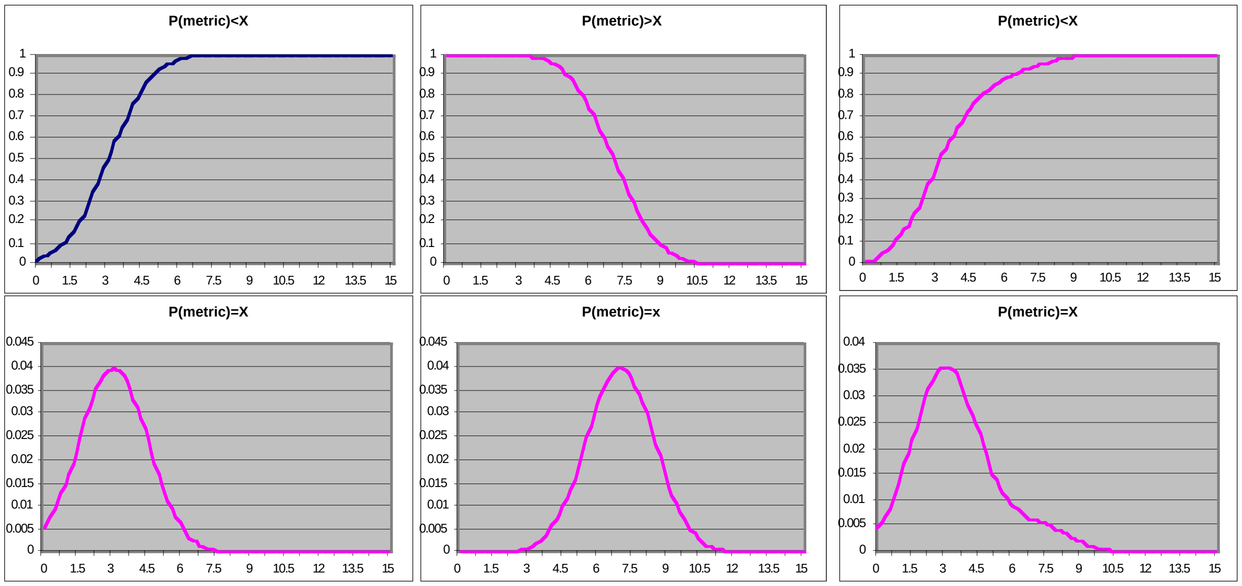

P(metric)=x
0 0.005 0.01 0.015 0.02 0.025 0.03
0 100 200 300 400 500 600 700 800 900 1000


P(metric)=X
0 0.01 0.02 0.03 0.04 0.05 0.06
0 100 200 300 400 500 600 700 800 900 1000

Name:	**Rev., Insp, & Wthroughs C**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation
Phase that it tells about:	Implementation
Meaning:	Measures the percentage of different types of V&V methods used in the implementation phase.
Modeled Range:	0 to 1
Use:	Evidence of V&V process quality
Comments:	
Reference:	Moller 1993, Freedman 1990, Redmill 1988Name:Name on model



Name:	**Cyclomatic Complexity**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Detailed Design
Phase that it tells about:	Implementation
Meaning:	Describes module complexity in terms of the number of nodes within a module that can transfer control to more than one node. For this model this measure is taken for the most complex module.
Modeled Range:	0 to 15. Ten represents the maximum ideal complexity
Use:	Used as evidence of the actual system complexity.
Reference:	IEEE 982



Name:	**Complexity C**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Complexity of the code. The node models three states: 1) Low – Coding should not pose challenges even for a development organization that would rate poorly against BTP-14. 2) Medium – Coding may pose a challenge to a poor development team, but should be within the capability of a development organization that rates well against BTP-14. 3) High –Coding may be beyond the capability of a development organization that rates well against BTP-14.
Modeled Range:	N/A
Use:	Influences the quality required of the development process. Influences the quality required of the V&V process.
Comments:	
Reference:	

Probability Table

Low	0.33
Medium	0.33
High	0.33

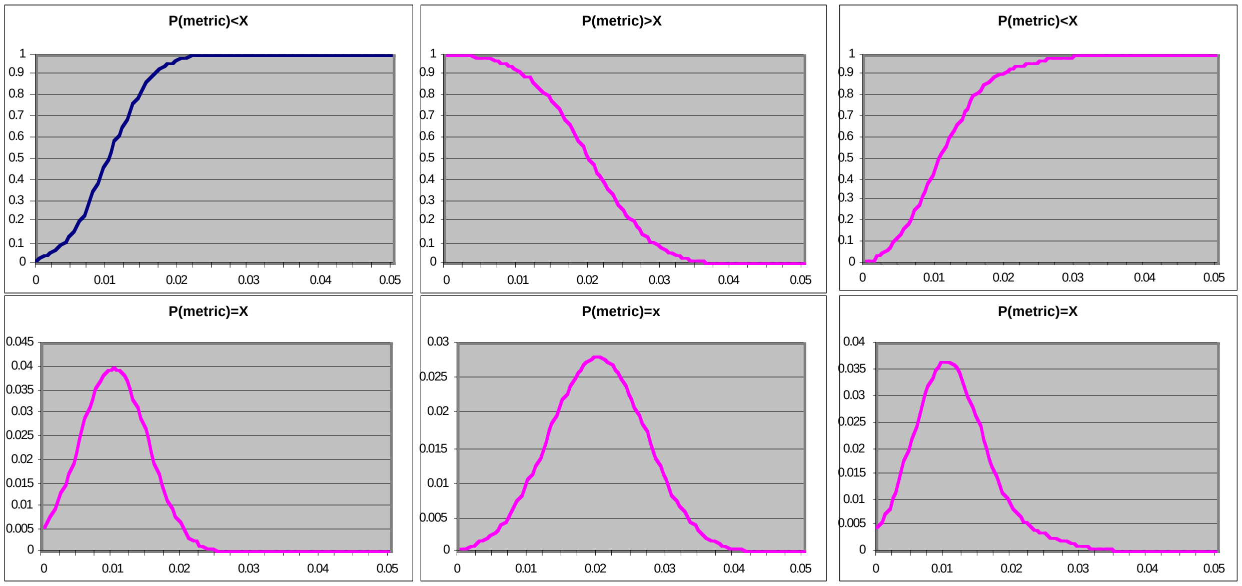
Name:	**Code Safety**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Architecture
Meaning:	Quality of the architecture with respect to the fundamental plant safety requirements. The node models two states: 1) Good – All fundamental plant safety requirements are completely, and correctly addressed in the code. 2) Poor – One or more fundamental plant safety requirements omitted or incorrectly addressed in the code.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences safety of the software design.
Comments:	
Reference:	

Probability Table

Requirements Safety	Correct		Incorrect	
Final Architecture	Good	Poor	Good	Poor
Safe	1	0	0	0
Not Safe	0	0	0	0

Anomaly Resolution Model

Name:	**Initial Defect Density AR**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Anomaly Resolution
Phase that it tells about:	Anomaly Resolution
Meaning:	The average of the total number of defects per line of code and the total number of defects per line of design statements discovered in the revisions made to address anomalies. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	0 to 0.05. Initial design or code densities greater than one per 20 lines probably indicates a failed development process.
Use:	Evidence of the quality of the initial revised design and code.
Comments	This measure is often expressed in defects per thousand lines instead of the defects per line expressed here.
Reference:	IEEE 982



Name:	**Initial Revisions**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Quality of the initial revision to requirements, architecture, design, or code to address anomalies found during validation. The node models two states:

1) Good – All errors in the initial revisions are detectable by a good V&V process and are correctable by a good development process. The implication is that the errors are not too numerous to be detected and corrected. It may also be important to consider the possibility that errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors.

2) Poor – The initial revisions contain some errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1.

The probability of the initial revisions being in these states is conditioned by the quality of the development process.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.

Use: Influences the quality of the final revisions.

Evidence of the quality of the development process.

Comments:

Reference: IEEE 982

Probability Table

Development Process	Good	Poor
Good	0.99	0.9
Poor	0.01	0.1

Name:	**Development Process AR**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the development process for revising requirements, architecture, design, or code to address anomalies discovered during validation testing. The node models two states: 1) Good – The revision process is good enough that it does not introduce safety-significant errors that cannot be detected by a good V&V process and that it can correct detected errors without introducing new safety-significant errors. The implication is that the development process does not introduce errors that are too numerous to be detected and corrected. It may also be important to consider the possibility that the process introduces errors are too subtle to be detected by the V&V process. This is not modeled because there are no metrics to give evidence for the subtlety of errors. 2) Poor – The revision process is expected to produce errors that cannot be detected by a good V&V process or corrected by a good development process. State 2 is the complement of state 1. The probability of the revision process being in these states is conditioned by the complexity of the code. More complex code is likely to require a better development process to produce adequate initial and final design.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the quality of the initial and final revisions.
Comments:	
Reference:	

Probability Table

Good	0.9
Poor	0.1

Name:	**CMM Level AR**
Type:	Discrete
Observable:	Yes
Phase where Measurable:	All
Phase that it tells about:	Anomaly Resolution
Meaning:	Software Capability Maturity Model measure for the anomaly resolution phase. Measures the predictability, effectiveness, and control of a project software processes. Two states are modeled: 1) CMM ≥ Level 3 — The software development process is defined, documented, and standardized. 2) CMM < Level 3 — The software development process is not defined. Success may still be possible based upon the quality of individual efforts or informal repetition of past successful practices.
Modeled Range:	Probability of state 0 to 1.
Use:	Evidence of the quality of the development process. Evidence of the quality of the V&V process.
Comments:	CMM typically describes the quality of the overall process regardless of phase. The model, however, includes the CMM measure separately for each phase. This allows for the possibility that the process maturity may change (hopefully improve) during the development process. This assumption also simplifies the modeling. The model could easily separately consider each of the five states described by the CMM measure. Currently, no significant benefit is seen from finer modeling.
Reference:	Paulk, Curtis, Chrissis, and Weber 2/1993, Paulk, Weber, Garcia, Chrissis, and Bush 1993, Paulk, Curtis, Chrissis, and Weber 7/1993

Probability Table

V&V	Good		Poor	
Development Process	Good	Poor	Good	Poor
< Level 3	0.01	0.95	0.95	0.99
≥ Level 3	0.99	0.05	0.05	0.01

Name:	**V&V AR**
Type:	Discrete – Prior Belief
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A

Meaning: Quality of the anomaly resolution verification and validation process. The node models two states:

1) Good – The anomaly resolution V&V process is good enough to detect all safety-significant errors in reasonably good initial code. The implication is that the V&V process can detect all errors they are not too numerous. It may also be important to consider the skill of the V&V team with respect to the development team. The V&V team must be capable of thoroughly understanding the development team's products. This is not modeled because there are no metrics to give evidence for the V&V team's skill.

2) Poor – The V&V process is not good enough to detect all safety-significant errors in reasonably good initial code. State 2 is the complement of state 1.

   The probability of the detailed design V&V process being in these states is conditioned by the complexity of the code. More complex code is likely to require a better V&V to detect errors in the initial.

Modeled Range: Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed V&V process.

Use: Influences the quality of the final code.

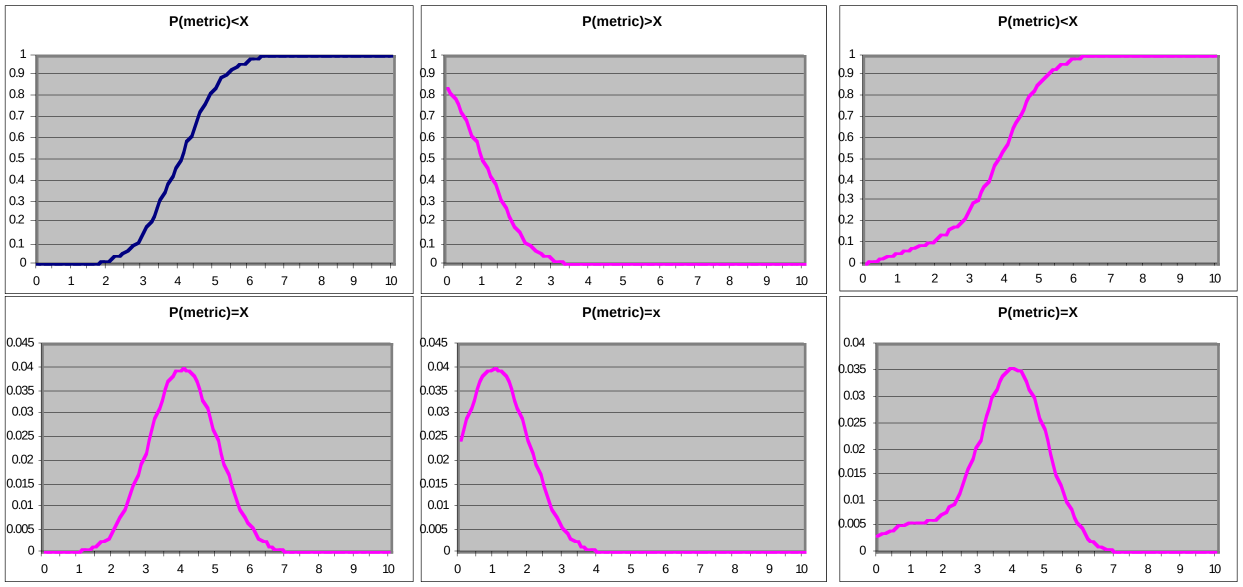
Comments:

Reference:

Probability Table

Good	0.9
Poor	0.1

Name:	**mh/Major Defect AR**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation
Phase that it tells about:	Implementation
Meaning:	Man-hours or V&V effort per major defect detected in the revisions to requirements, architecture, design, or code during the anomaly resolution process. The measure is applied to new code development. A low number (less than 3 hours) indicates potential problems with the requirements. A high number (greater than 5 hours) indicates potential problems with the V&V process.
Modeled Range:	Man-hours between major defect between 0 and 10 hours. A measure greater than 10 indicates either an extremely good initial design document or a failed V&V process. If it can be determined that the former is the case, the state of the architecture may be directly entered into the model, thus bypassing this measure.
Use:	Evidence of the quality of the initial revisions made to resolve anomalies detected during validation. Evidence of the quality of the code V&V process
Comments:	
Reference:	IEEE 982

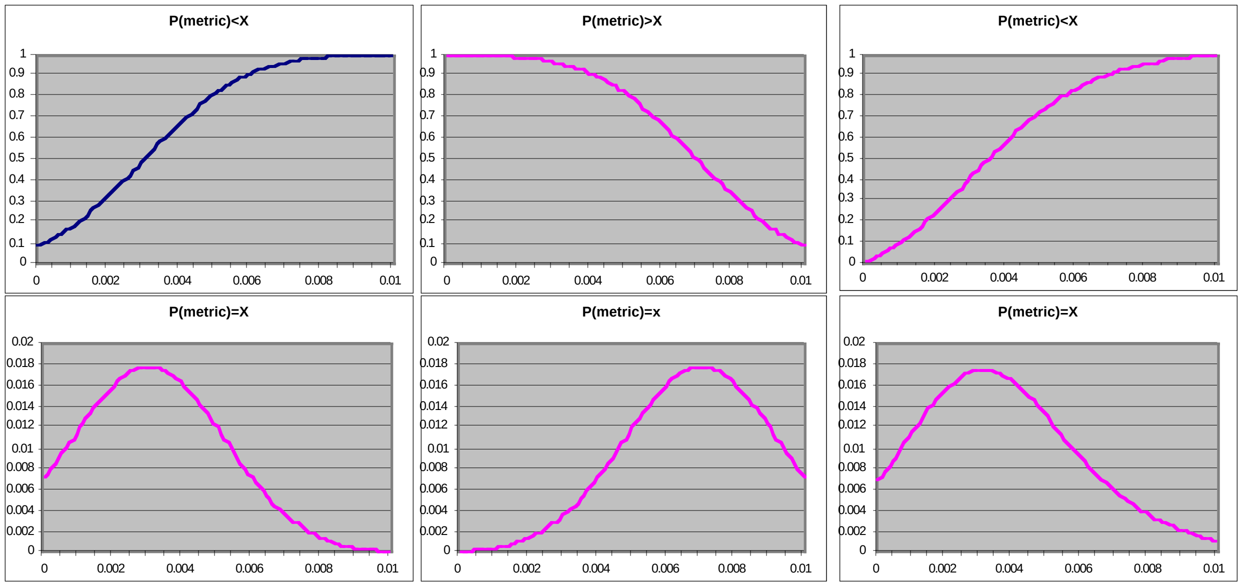


Name:	**Final Revisions**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the final revisions made during anomaly resolution. The node models two states: 1) Good – All safety-significant findings of the validation process are completely, and correctly addressed in the final revisions to the requirements, architecture, design, and code. 2) Poor – One or more safety-significant findings of the validation process are not completely, and correctly addressed in the final revisions to the requirements, architecture, design, and code.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences the safety of the final code. Evidence of the quality of the development process. Evidence of the quality of the V&V process. Evidence of the quality of the initial revisions.
Comments:	
Reference:	IEEE 982

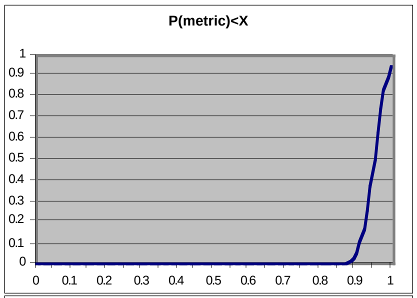
Probability Table

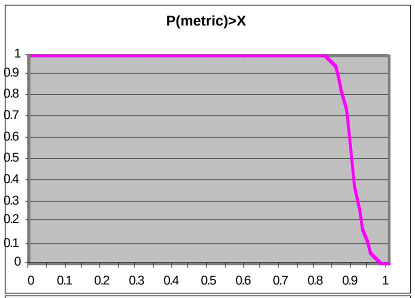
Initial Revisions	Good				Poor			
Development Process	Good		Poor		Good		Poor	
V&V	Good	Poor	Good	Poor	Good	Poor	Good	Poor
Good	0.98	0.89	0.69	0.49	0.89	0.81	0.63	0.45
Poor	0.02	0.11	0.31	0.51	0.11	0.19	0.37	0.55

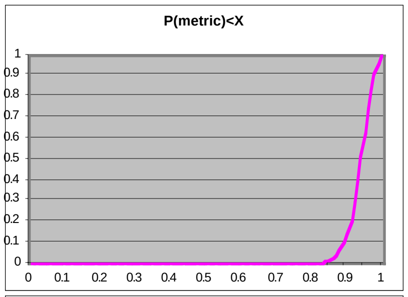
Name:	**Final Defect Density AR**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Implementation
Phase that it tells about:	Implementation
Meaning:	The average of the total number of defects per line of code and the total number of defects per line of design statements discovered in the revisions made to address anomalies. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the design.
Modeled Range:	0 to 0.01. Final defect densities greater than one per 100 lines probably indicates a failed development process.
Use:	Evidence of the quality of the final code.
Comments	This measure is often expressed in defects per thousand lines instead of the defects per line expressed here.
Reference:	IEEE 982

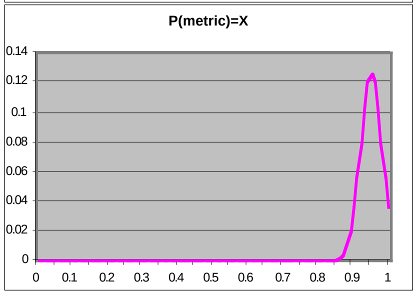


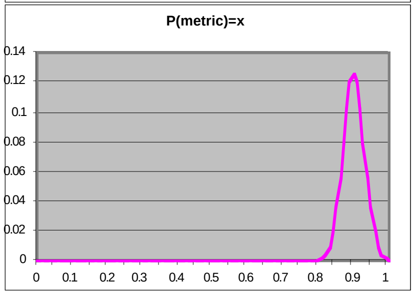
Name:	**Initial FPA AR**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Anomaly Resolution
Phase that it tells about:	Anomaly Resolution
Meaning:	Function point analysis quality measure on the initial draft of revisions to the requirements or architecture. For the anomaly resolution phase the total number of errors in both the architecture and requirements revisions is used. The quality figure is expressed as 1 minus defect density per function point. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the requirements.
Modeled Range:	FPA 0.8 to 1. An initial FPA < 0.8 probably indicates a failed development process.
Use:	Evidence of the quality of the initial requirements, architecture, design, and code.
Comments	
Reference:	IEEE 982

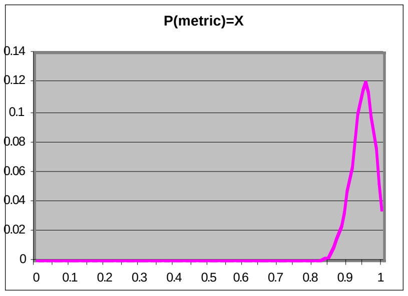




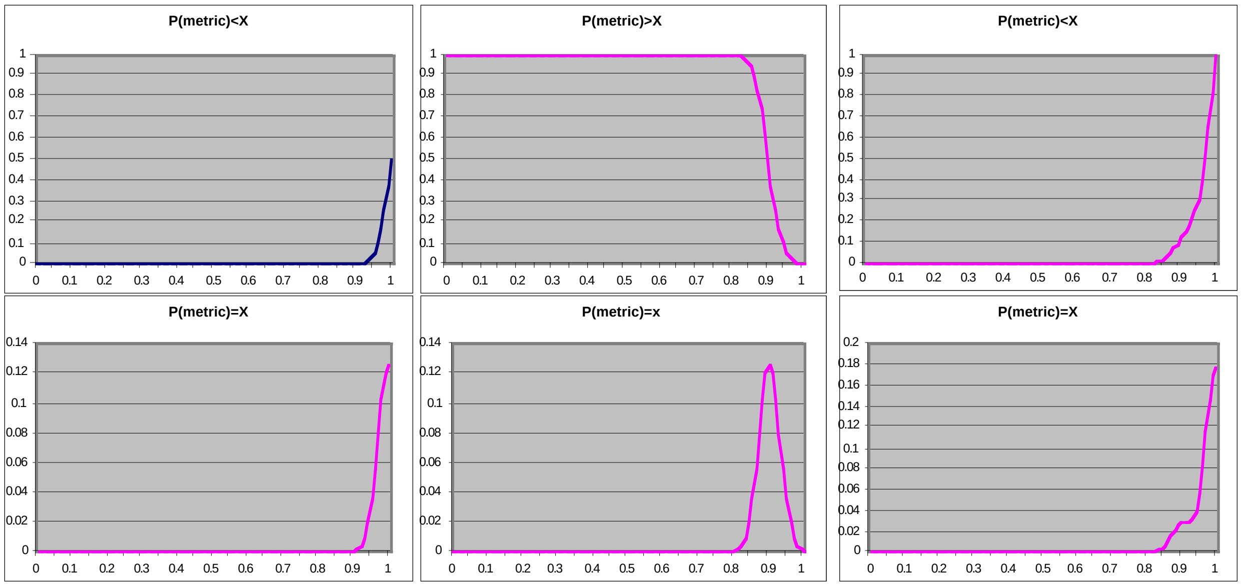






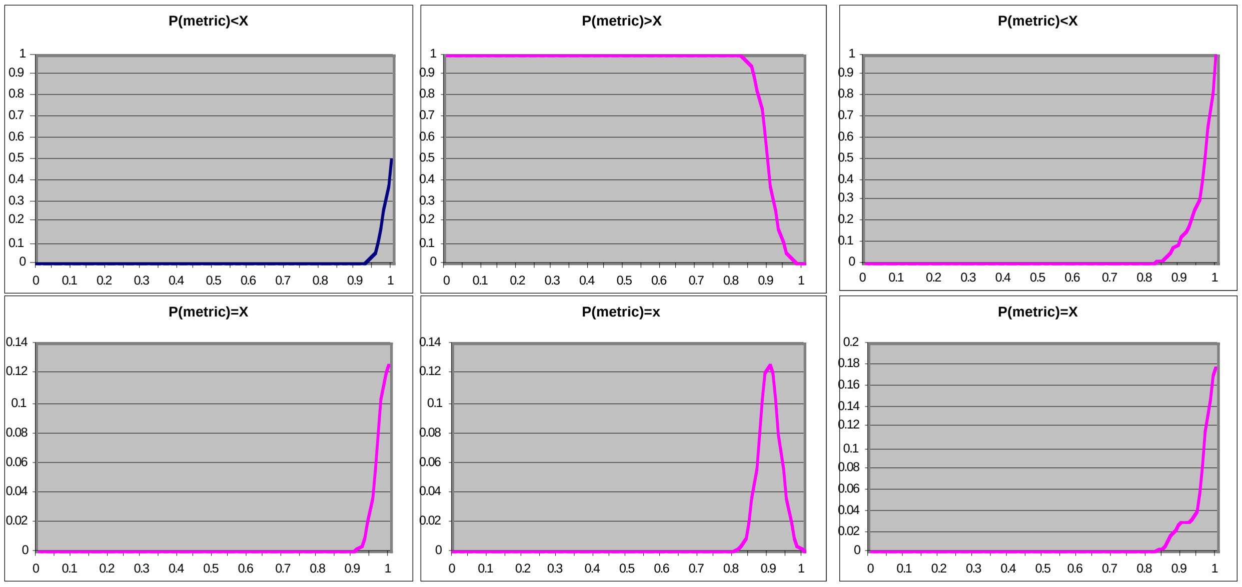


Name:	**Final FPA AR**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Anomaly Resolution
Phase that it tells about:	Anomaly Resolution
Meaning:	Function point analysis quality measure on the final revisions to the requirements or architecture. For the anomaly resolution phase the total number of errors in both the architecture and requirements revisions is used. The quality figure is expressed as 1 minus defect density per function point. It is assumed that both safety-significant and non-safety-significant errors are counted as they are both indicators of the overall quality of the requirements.
Modeled Range:	FPA 0.9 to 1. An initial FPA < 0.9 probably indicates a failed development process.
Use:	Evidence of the quality of the initial requirements, architecture, design, and code.
Comments	
Reference:	IEEE 982



Name:	**Reliability Growth**
Type:	Discrete
Observable:	Yes
Phase where Measurable:	Anomaly Resolution
Phase that it tells about:	Anomaly Resolution
Meaning:	The reliability growth parameter. Two states are modeled: 1) Positive – The anomaly resolution process is removing more errors than it introduces. 2) Negative – The anomaly resolution process is inserting more errors than it removes.
Modeled Range:	N/A
Use:	Evidence of the quality of the anomaly resolution process.
Comments:	
Reference:	IEEE 982

Probability Table



Name:	**Complexity AR**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Composite complexity of the software. The node models three states: 1) Low – Anomaly resolution should not pose challenges even for a development organization that would rate poorly against BTP-14. 2) Medium – Anomaly resolution may pose a challenge to a poor development team, but should be within the capability of a development organization that rates well against BTP-14. 3) High – Anomaly resolution may be beyond the capability of a development organization that rates well against BTP-14.
Modeled Range:	N/A
Use:	Influences the quality required of the development process. Influences the quality required of the V&V process.
Comments:	
Reference:	

Probability Table

Low	0.33
Medium	0.33
High	0.33

Name:	**Anomaly Resolution**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Architecture
Meaning:	Quality of the anomaly detection and resolution with respect to the fundamental plant safety requirements. The node models two states: 1) Good – All residual errors relating to fundamental plant safety requirements are were identified by testing and completely, and correctly addressed by the anomaly resolution process. 2) Poor – One or more residual errors relating to the fundamental plant safety requirements were not detected during testing or incorrectly addressed in the anomaly resolution process.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences safety of the software design.
Comments:	
Reference:	

Probability Table

Testing	Correct		Incorrect	
Final Revisions	Good	Poor	Good	Poor
Good	1	0	0	0
Poor	0	0	0	0

Name: **Cyclomatic Complexity C**

This is the same node as used in the coding phase. It is shown as a separate node for clarity, but in the actual model only one node is used.

Name: **System Design Complexity**

This is the same node as used in the detailed phase. It is shown as a separate node for clarity, but in the actual model only one node is used.

Name: **Static Arch Complexity**

This is the same node as used in the architectural design phase. It is shown as a separate node for clarity, but in the actual model only one node is used.

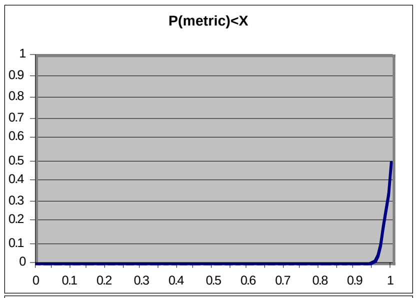
Validation Testing Model

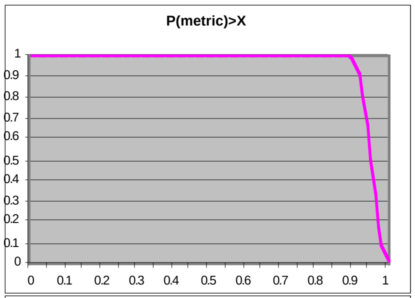
Name:	**Testing**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	N/A
Meaning:	Quality of the software testing. The node models two states: 1) Good – Testing is adequate to identify all residual safety related errors in the final code. 2) Poor – One or more residual safety-significant errors are not detected by testing.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed test process.
Use:	Influence on test quality measures. Influence on anomaly resolution safety.
Comments:	
Reference:	IEEE 982

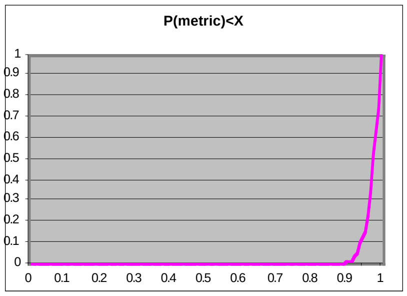
Probability Table

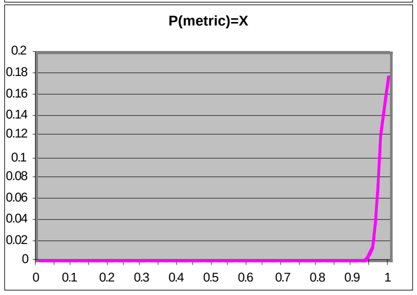
Good	0.9
Poor	0.1

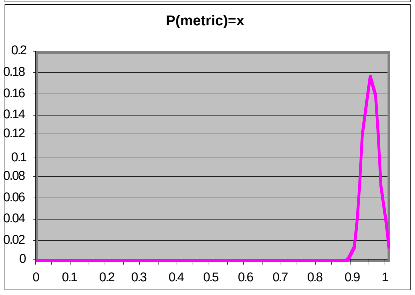
Name:	**Modular Test Coverage**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Testing
Phase that it tells about:	Testing
Meaning:	The percentage of modules for which all test cases have been satisfactorily completed.
Modeled Range:	0.9 to 1. Probabilities < 0.9 probably indicate a failed test process.
Use:	Evidence of the quality of testing.
Comments:	
Reference:	IEEE 982

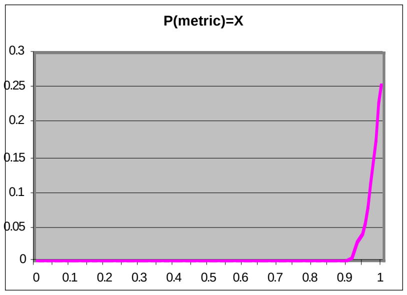




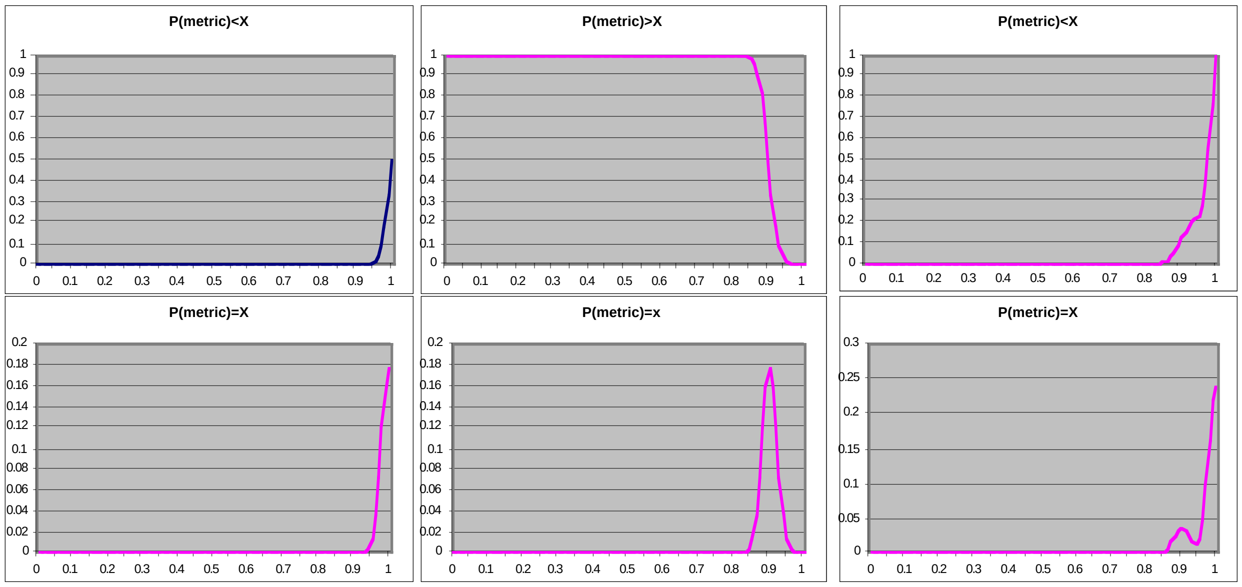




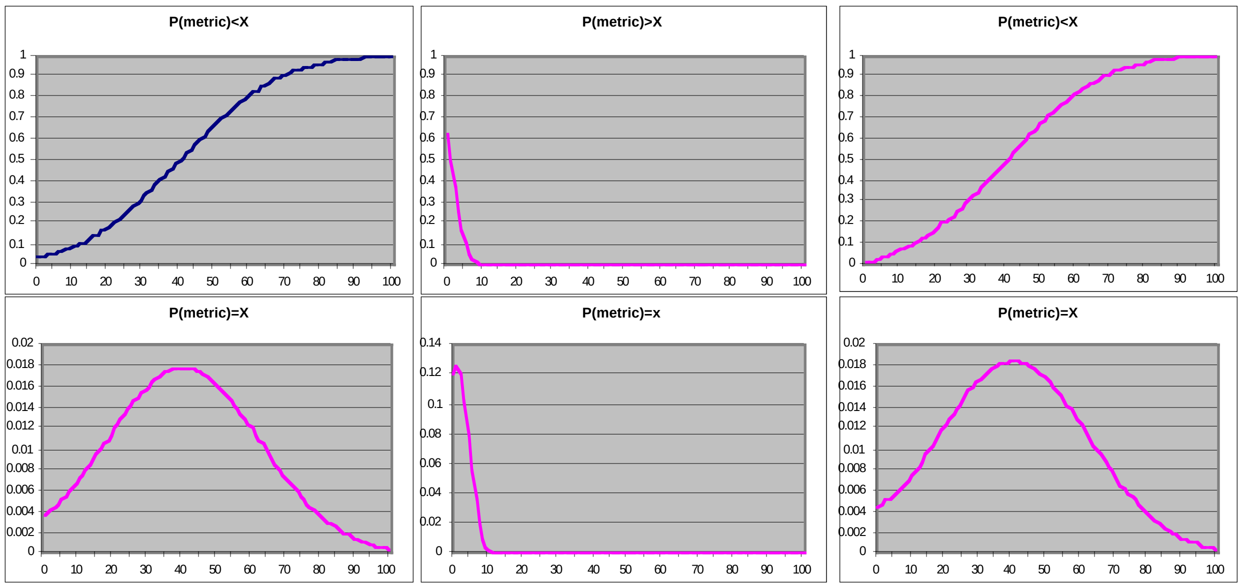




Name:	**Functional Test Coverage**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Testing
Phase that it tells about:	Testing
Meaning:	The percentage of functional requirements for which all test cases have been satisfactorily completed.
Modeled Range:	0.9 to 1. Probabilities < 0.9 probably indicate a failed test process.
Use:	Evidence of the quality of testing.
Comments:	
Reference:	IEEE 982



Name:	**MTT Next Fault**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Testing
Phase that it tells about:	Testing
Meaning:	Mean Time to Discover Next Fault at the completion of the statistical test program. Indicates the degree of completeness of the statistical testing. Testing should continue until this time is much greater than the desired reliability.
Modeled Range:	1000 to 100,000 hours. Statistical testing to establish mean time to next fault greater than 100,000 hours is probably not practical. Testing that terminated when the mean time to next failure is predicted to be less than 1000 hours indicates a failed test program.
Use:	Evidence of test quality.
Comments:	
Reference:	IEEE 982



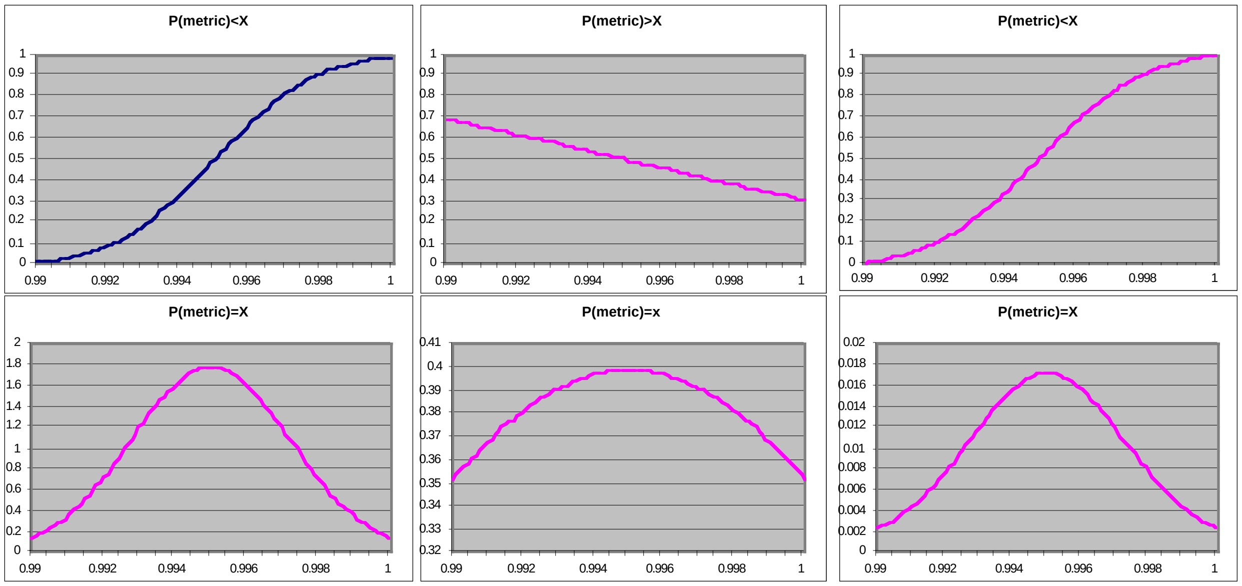
Final Product Model

Name:	**Validated Software Safety**
Type:	Discrete – Conditional Probability Table
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Final Product
Meaning:	Quality of the final software with respect to the fundamental plant safety requirements. The node models two states: 1) Good – The delivered software completely, and correctly addressed by the anomaly resolution process. 2) Poor – One or more residual errors relating to the fundamental plant safety requirements remain in the delivered software.
Modeled Range:	Probability that state is good 0.9 to 1. Probabilities < 0.9 probably indicate a failed development process.
Use:	Influences software reliability.
Comments:	
Reference:	

Probability Table

Code Safety	Safe		Not Safe	
Anomaly Resolution	Good	Poor	Good	Poor
Safe	1	1	0.9	0
Not Safe	0	0	0.1	1

Name:	**Observed Reliability**
Type:	Continuous
Observable:	Yes
Phase where Measurable:	Testing
Phase that it tells about:	Final Product
Meaning:	The software reliability measured by statistical testing or the final product. One of several measures such as Failure Rate, Run Reliability may be used to determine this measure. A number of models are available to convert observed failure rates into an estimate of observed reliability. Methods for analyzing predictive accuracy (e.g., prequential likelihood ratio, u-plots, y-plots) may be used to select the best estimate for observed reliability. For complicated software or systems the observed reliability may be calculated by measuring reliability for individual components and estimating the overall reliability using modeling techniques such as reliability block diagrams, Markov models, or input domain modeling.
Modeled Range:	0.99 to 1. Observed reliability less than 0.99 indicates a failed development process.
Use:	Evidence of software reliability Evidence of test quality
Comments:	
Reference:	



Name:	**Reliability**
Type:	Discrete — Prior estimate
Observable:	No
Phase where Measurable:	N/A
Phase that it tells about:	Final Product
Meaning:	The predicted software reliability presented as a discrete distribution function estimating the probability that the software reliability is < 0.99, between 0.99 and 0.999, between 0.999 and .9999, or > 0.9999.
Modeled Range:	0 to 1
Use:	Model output.
Comments:	The prior value should be adjusted based upon some initial estimate of software reliability. Initial estimates might come from measures such as Project Initiation Reliability Prediction or the Gaffney Bugs per Line of Code measure. Both of these measures estimate fault densities. The estimated fault density must be converted to a reliability estimate for input to the model. It is important the prior estimates cover the full range of reliability distribution modeled. Entering a prior estimate of zero for any part of the prior distribution will force to zero the final probability that the reliability is within that range.
Reference:	

Probability Table

Validated Software Safety	Safe	Not Safe
< 0.99	0.05	0.5
0.99 – 0.999	0.15	0.3
0.999 – 0.9999	0.5	0.15
> 0.9999	0.3	0.05

APPENDIX B
DETAILS OF THE COMBINATORIAL MODEL

This appendix describes the quality functions for the combinatorial model. Each function is described by 5 parameters: a shape identified, the practical range of the measure (2 parameters) and inflection points (2 parameters). The quality functions relate each measure to the expected change in defect density form the average defect density observed over the class of process system software.

The shape factor reflects the relationship between defect density and measure. For example: for mean time to next failure measure, as the mean time to next failure increases, the quality of the testing is better so the defect rate decreases, hence a Z shape. For the man-hours per major defect measure, as the measure increases the it is inferred that the quality of the V&V process is lower, hence the defect rate increases Note, this relationship reverses when mh/md is used as a product quality measure.

The practical range indicate the points at which further increases or decreases in the measure give no additional information about the expected defect density. These are essentially the points at which the measure value leads to the conclusion that the modeling assumptions discussed in section 2 are violated. For example, a very low mean time to next failure would indicate that the defect rate is so high that the software under consideration violates the assumption that the model is dealing with software of reasonably good quality.

The inflection points define the points at which changes to the measures only have a weak influence on the expected defect density.

Table B.1 Process quality factor input metrics

Measures	Shape	Practical Range	Inflection Points	Notes
Mean Time to Next Failure	Z	4-1000	10 - 60	Inflection points based upon RADC information on typical fault densities and failure rate – fault density conversion
Man-hour per major defect in requirements and design	S	0-10	3, 5	Inflection points based upon typical values provided in IEEE 982.1
Man-hours per major defect in implementation	S	0-10	3, 5	Inflection points based upon typical values provided in IEEE 982.1
Modular test coverage	Z	0-1	0.9, 1	Inflection points based upon belief that development processes in accordance with BTP-14 should achieve a very high level of test coverage.
Reviews, Inspections, and walkthroughs	Z	0-1	0.5, 0.8	Inflection points based upon belief that there is a relatively high degree of overlap between various review methods.
Capability Maturity Model	Z	2-5	2, 3	Inflection points based upon belief that a development process in accordance with BTP-14 will normally be rated at least at level 3. Level 2 processes might be acceptable under some circumstances and Level 1 processes are unacceptable. Level 4 and 5 represent the extensive ness of the findings to other areas of operation, thus this information gives little additional information about a specific project.
Fault number phases	Z	0-6	0.5, 2	
Requirements change requests per requirement	S	0-2	0.1, 0.5	

Table B.2 Requirements and design quality factor input metrics

Measures	Shape	Range	Inflection Points	
Cause & effect graphing	Z	0.8-1	0.85, 0.95	Inflection points based upon belief that a relatively small number of ambiguous requirements represents a significant problem, but that it is nearly impossible to remove all ambiguity in a natural language specification.
Design defect density	S	0.05-0.002	0.02 – 0.005	Inflection points based upon Rome Laboratory estimates of typical defect densities for quality software.
Man-hour per major defect in requirements and design	Z	0-10	5, 3	Inflection points based upon typical values provided in IEEE 982.1
Function point analysis for requirements	Z	0.8-1	0.9, 0.99	Inflection points based upon expectation that very complete requirements are necessary for high reliability software.
Function point analysis for design	Z	0.8-1	0.9, 0.99	Inflection points based upon expectation that very complete requirements are necessary for high reliability software.
Graph-theoretic static architecture	S	0-15	3-10	Inflection points based upon IEEE 982.2 discussion. Static architecture of 3 represents a simple tree architecture. There is little benefit to simplifying the design beyond this point.
Requirements traceability	Z	0.8-1	0.9, 0.99	Inflection points based upon expectation that very complete design necessary for high reliability software.

Table B.3 Implementation quality factor input metrics

Measures	Shape	Range	Inflection Points	
Code defect density	S	0.05-0.002	0.02 – 0.005	Inflection points based upon Rome Laboratory estimates of typical defect densities for quality software.
Man-hour per major defect	Z	0-10	5, 3	Inflection points based upon typical values provided in IEEE 982.1
Cyclomatic Complexity	S	0-15	3-10	Inflection points based upon IEEE 982.2 discussion. Static architecture of 3 represents a simple tree architecture. There is little benefit to simplifying the design beyond this point.