Technical Report No: 2003/12

A model for inspection efficiency prediction

T.Cockram

P.A.V.Hall

D.C.Ince

7th November 2003

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

A model for inspection efficiency prediction

T. Cockram, P.A.V. Hall and D.C.Ince
Department of Computer Science,Open University, Milton Keynes,United Kingdom. MK76AA
D.C.Ince@open.ac.uk

Abstract

While inspections are a valuable tool for software quality assurance, inspection models are labour intensive, require knowledge of all errors in a software product, make questionable assumptions, and do not capture the experience of inspectors. In this paper we describe a novel inspection model based on Bayesian belief networks that overcomes many of these problems. We describe the problems which affect the inspection process, outline how Bayesian belief network are able to provide a powerful mechanism and describe data taken from a large number of inspections which provide a validation of the model.

1. Introduction

A software project manager is faced with a number of tricky decisions regarding software inspections, for example given limited resources what members of staff should be allocated to the inspection process given their different levels of experience. An effective inspection model which aids this decision making would clearly be a major tool in the armoury of the project manager.

This paper describes such a model. We provide a brief description of a new model whose novelty arises from the fact that it overcomes exiting problems with current models. Firstly, we briefly look at software inspections, then we describe the problems with exiting software inspection models. Our model is based on an technology known as Bayesian belief networks. The paper describes such networks and details how they can be applied to the inspection process. Finally the validation of the model on a major industrial project is detailed.

2. Software Inspections

The IEEE glossary of software engineering [IEEE 88] provides the following definition of a software inspection:

> *A formal evaluation technique in which software requirements, design or code are examined in detail by a person or group other than the author detecting faults, violations of development standards and other problems.*

Software inspections were first described in Fagan's classic paper [Fagan76] which still provides the basis of much software inspection practice. This paper described the need for improving methods for ensuring quality in the production of software and described how inspections are a formal, efficient and economical method of finding errors.

Fagan's work follows industrial principals of statistical process control in order to make improvements in the quality of the software produced. Software inspections are a technique where the software is examined in a formal process with a clearly defined series of operations to identify errors. These inspections are conducted by a number of people with defined roles such as moderator, designer, coder and tester.

Experience with software inspections has been excellent with the vast majority of those studies which carried out statistical analyses reporting improvement. For example Reeve [Reeve91] applied Fagan inspections at MEL as part of an ISO 9001 process and noted a 1 to 90 cost saving between hours used for inspection and hours saved in subsequent rework. Doolan [Doolan92] applied inspections to software supporting seismic investigations. He found that by using Pareto analysis, the application of inspections during the early stages of a project gave the most benefit at the least cost. Productivity figures of 1 to 30 hours saved were noted. Bush [Bush90] and Kelly [Kelly92] have reported on the use of formal software inspections as technical review processes within the NASA Jet Propulsion Laboratories. Bush noted that estimated savings of $1595 per were obtained.

The technical literature clearly supports inspections as an effective quality assurance mechanism. However, there are still some important problems remaining.

3. Problems

3.1 Knowledge of all errors

Current inspection effectiveness models require knowledge of *all* errors in a product. All the published inspection effectiveness models require knowledge of the total number of errors associated with a product or the number of errors remaining in a product after inspection and correction. Fagan makes use of these quantities within his equation of effectiveness. Unfortunately these quantities are not available at the time of inspection, and may never be known unless the execution of the software cause all of the errors in the software to be manifested as faults. What is needed is a model of inspection effectiveness that does not require knowledge of the number of errors. Ideally, such a model should be employed as a predictive mechanism prior to an inspection as a management aid to ensure that the resources are available and to provide assistance in making the inspection as effective as possible. This paper describes such a model.

3.2 Questionable assumptions

Published inspection effectiveness models, for example [Chris88, Chris90, Porter88] all make major assumptions about the nature of software inspections and the way in which errors are distributed. For example, Christenson's model of code inspections assumes that the density of errors is proportional to the density of problem reports raised. Porter *et al*. [Porter88] make a similar assumption but use a generalised linear model rather than the straight-line approach of Christenson. This assumption is unreasonable, as the absence of errors detected does not indicate freedom from errors, only that the process has failed to find them. Counting the numbers of errors identified in an inspection is not sufficient. Say we found X errors, then this could represent all the errors in a product, resulting from a good product and a good review process. The same number of errors X, however, could be only a fraction of the total number of errors, resulting from a poor inspection process applied to a bad product. In fact many combinations of product and inspection process could result in X errors being discovered. Porter's model also assumes that the process of making errors in the code is a random process. This implies that the error density over different projects is constant. This could only be possible if it is assumed that there are many different sources of error. A similar mechanism occurs in mechanical reliability; however, as the causes of error are eliminated and defects repaired, as in the case of a software development process, non- random distributions become important.

With software certain types of error can dominate e.g. requirement errors [Lutz93]. This random distribution is only true when the different projects are directly comparable, for example, the same software development team, a similar sized project etc. Many models also combine data from a number of disparate sources in an uncontrolled way, without considering the dependencies between the sources of information.

The model of inspection effectiveness proposed his paper, does not make any assumption about the density of errors within a product. It is based only on the disparate attributes and expert opinion of the dependencies between attributes that contribute towards inspection effectiveness.

3.3 Capturing the experience of inspectors

Clearly the experience of inspectors is an important attribute in the effectiveness of an inspection. Current models and experiments, for example Porter *et al.* [Porter94] attempt to eliminate the effect of inspectors learning from the inspection process. While this assists with conducting clean experiments it does not reflect reality. Inspections are effective because of the lessons learned, so any model of effectiveness should include this learning process and be adapted as experience in conducting inspections grows. Current literature has not addressed the potential for the use of technologies such as those associated with artificial intelligence in improving the effectiveness of software inspections. The model of software inspection effectiveness detailed here makes use of previous experience and expert judgement, and investigates the potential for the model to learn with experience.

4 Bayesian Belief Networks

Bayesian Belief Networks are based on Graphical Probability Models (GPM) and use the structure of a GPM together with a numerical model of the dependencies between the attributes to calculate the probability of a property having a certain state or range of states using Bayesian statistics. The numerical model of dependencies can be used as a means of providing *a-priori* experience to the model.

Graphical probability models are based on the idea of mapping causal relationships within a network. This approach was first described by [Wright21, Wright 34].

These ideas were extended to combine probabilistic methods in an expert system using causal networks by Pearl [Pearl88] in which he introduced the idea of procedural semantics based on a causal model of a set of variables *U* in a directed acyclic graph (DAG) in which each node corresponds to a distinct element of *U* with arcs showing inferred causation. A variable *X* is said to have a causal influence on a variable *Y* if a strictly directed path from *X* to *Y* exists in every minimal causal model consistent with the data.

The problem is represented by the causal structure, with prior belief asserted for each leaf node within a tree structure; application of Bayes theorem allows the inference for each junction within the tree to be updated. Bayes Theorem states that the probability of A, given that B is known, is equal to the probability of B given that we know A, multiplied by the ratio of probabilities A and B.

$$P(A\&B) = P(A|B)\,P(B) = P(B|A)\,P(A)$$

A causal net consists of a number of nodes, which represent variables, which can take discrete values (or a continuous distribution, which can be approximated by a series of discrete values over the range of the distribution), linked together. The links represent the dependencies between the variables. Each link represents the influence which the value of one variable has on the value of another (if the influence is zero, then there is no link). The influence depends on the combination of nodes: in general if one node can take *n* values and the other *m* values, the influence of one node on the other is an n x m matrix.

For example, Figure 1 describes a net for a quality of inspection procedure. In designing this part of the network it has been decided that this depends on formal actions, adequate inspection rate, defined evaluation criteria and a defined scope of inspection. These are shown by the directed links on the diagram, from the child nodes to the parent. These parent nodes within the network are described as evidence nodes for which data is provided and which, combined with a model of dependencies between these attributes, estimates can be calculated. These evidence nodes are required to be conditionally independent from each other. It is necessary to select metrics for evidence nodes that are orthogonal. The network can also have a number of nodes that link evidence nodes and may be used to describe a hierarchy within the model. These nodes are therefore conditionally dependent on the inputs and therefore the relationship between the inputs must be described. The relationship between these attributes is determined from past experience (expert judgement) and is known as the prior belief.

In the DAG shown in Figure 2 above, the value of *Z* depends on two inputs *X* and *Y.* If the value of *Z* is not known but if we have evidence for the value of X and that the value of *X* has no influence on the value of *Y*, then *X* and *Y* are conditionally independent. If however the value of *Y* is influenced by the value of *X* then the inputs *X* and *Y* are not independent. Additionally if *Z* has its own data associated with it then, the values for *X* and *Y* are conditionally dependent on *Z* to satisfy it. Therefore data used to provide evidence for *X* and *Y* must be separately determined.

Bayesian belief network work is now mature, for example there are a number of tools that have been developed to implement Bayesian networks, of which HUGIN [Hugin98] of the commercial products is the best known. Other commercial products are Analytica [Morgan98] which has been developed for supporting automated decision systems using influence trees and Ergo [Herskovitz97] for medical diagnosis.

Bayesian networks have been successfully applied in several domains such as medical diagnosis [Andreassen87, Herskovits97] where much of the early work has been carried out. Further applications include the assessment of damage to structures in civil engineering [Reed93] reliability estimation [Shaw91], system reliability [Littlewood98]. A recent use which many computer users will be familiar with is for providing intelligent help in Microsoft Office 97 [Horvitz98] where the active “paper-clip help icon” invokes a Bayesian search engine that given the query provided by the users the network finds the most likely topics to support the query.

5. Belief Networks and Inspections

A network that represents the attributes that influence good software inspection effectiveness was developed; it was partly based on Fagan's original model augmented by the experience of the authors and factors detailed in the software engineering literature. Working from the objective, a software inspection network was built down to the measurable attributes that will be used in the Bayesian model. A high level view of the network is shown as Figure 3. This network was then used by a Bayesian inference engine.

For each leaf node in the network metrics were defined, these were; size of the item being inspected, complexity of the item being inspected, experience at inspection preparation, adequate preparation time, adequate inspection checklist, formal actions taken, adequate inspection rate, defined exit criteria, defined scope of inspection, team size, experience in inspection role, adequate application experience, adequate domain knowledge, training/experience at inspection and communication skills. Many of these metrics had simple yes/no values, for example the question 'Was there adequate preparation time? was used to quantify the preparation time metric. The size of item was characterized by the number of non-commented lines of code (NCLOC), the complexity of item by McCabe's complexity measure [McCabe76], the inspection rate was measured as NCLOC divided by inspection time and communication skill was judged as either being poor, fair or good.

The process of employing the model consisted of:

- Surveying staff opinion in order to set the initial values of the probabilities in the network.
- Defining the initial values for the network.
- Initializing the network with the prior beliefs of staff.
- Calibrating the network.
- Verifying the model using a variety of statistical techniques.

5.1 The project

The model was validated on a very large software project carried out by a consortium of companies who were developing a real-time system, although all the data used for the study was obtained from one of the partners in the consortium. The groups of companies on the case study project had been independently assessed as reaching at least level 3 SEI

The software developed was a condition monitoring system used with a gas turbine engine, software inspections and checklists were required to be to standards appropriate for this safety/integrity level. The research reported here covers the design, coding and testing phases of the project, with inspections conducted during the design and coding stages.

The errors identified in subsequent testing were used in providing evaluation evidence for the software inspection effectiveness model; this is covered in a later section of this paper.

The project was designed using the LVM-OOD technique and mostly coded in a safe subset of Ada, with eight modules coded in 68020 assembler. The code for the project consisted of 1261 code units of Ada package specifications and bodies, package instantiation of generic packages, task bodies, procedure bodies, function bodies and 68020 assembler subroutines.

The software inspection method used on the project was based on phased inspections described by Knight and Meyers [Knight91] This method uses the idea of inspectors logging issues independently, with a meeting only held when a difficult or contentious issue is raised [Votta93]. To provide consistency for the metrics collection and to identify basic errors a simple static code analysis tool was used [Hennel89].The inspectors appointed to inspect an author’s work were either other members of the project team or other experienced software engineers within the same department.

Fourteen inspectors were used of which 6 had been authors of modules for this project, In the case of the 6 who had been authors, the modules they inspected were not modules that they had been involved in writing. The inspector moderators were software quality controllers who are members of the quality department and therefore have an independent reporting route from the authors and inspectors. Three moderators were used for the project.

5.2 Initial data collection

A questionnaire that was used was one which had been validated and piloted on a British government funded project which investigated the maintenance of quality factors in safety-related software. Two questionnaires were issued: one for moderators and one for inspectors.

The inspectors' questionnaire contained both direct questions, which produced data to feed directly into the model and questions to validate some of the subjective answers provided by the inspectors. The moderators questionnaire focused on the role of the moderator and fed into the 'Quality of moderator' node of the model. Other questions about the product and process were included to be completed by the moderator. Some of the data was automatically collected by the static code analysis of the software, e.g. size and complexity.

To provide data for the model calibration and verification, data from the subsequent testing and rectification phases of the development life-cycle was required. The main source of this data was the problem report and correction system used by the project to monitor and record all changes to the software. As part of this system, an analysis of all problems and changes was conducted. This analysis identified the point of introduction.

5.3 Network initialisation

The inspection effectiveness model described in this paper required initialising by establishing the prior belief of the conditional probability distribution of intermediate variables. The initialisation of a Bayesian network requires that the *a priori* belief in terms of the conditional probability for each state of the variables in the parent nodes be specified. Experience is used to provide an *a priori* conditional probability value for each node. For the evidence nodes, which are at the bottom of the network, the initial distribution for each state of these variables was set to be flat over its range, i.e. the evidence had an equal probability for each state.

This experience was elicited by conducting a survey from two independent groups of engineers with experience of inspection. The two groups operated a similar inspection process at two different sites of the same organisation.

The results from the survey were then translated into the *a-prori* conditional probability tables that were required for the Bayesian Belief Network. These tables were completed using a brain storming activity which was facilitated with a subset of the inspectors surveyed. The inspectors completed the tables using the ranking determination as the guidance for completing all the values in the table.

5.4 Network calibration

When a network's probability predictions matches the actual frequency of occurrence within a given tolerance, the network is said to be calibrated [O'hagan94] The network was calibrated using metrics data from the case study being fed into the model employing a commercial Bayesian network tool [Hugin98]. The resulting distribution of software inspection effectiveness predicted by the model is compared with the actual software inspection effectiveness, which is found from the equation:

$$\frac{n_r}{n_t}$$

where the inputs for this equation are found from the inspection and post inspection data described in Section 5.2 and where n_r is the number of issues found at inspection and n_t the number of issues found during further inspection or testing. By using the Bayesian propagation methods the conditional dependency assignments for the intermediate nodes of the model were then adjusted. Data from the post inspection activities described above was used. As the calibration proceeded the error between the predicted result and the actual result reduced as the model learnt from the data being entered.

6 Model validation

The result of the calibration detailed in Section 5.4 is a model that can be used to predict the outcome of an inspection. A number of validations were carried out in order to judge the effectiveness of the model**.** The validation of most interest to readers of this paper are those which provide confirmation of the predictive ability of the model. This is detailed later in the paper. However, for completeness sake it is worth outlining some of the other validation activities that took place in Sections 6.1 and 6.2[1].

6.1 Initial testing

The purpose of initial testing is to provide a set of results for a benchmark against which the learning potential of the network can be compared. A sample of 100 inspections was taken, which represents about 8% of the total number of inspections from the project. These test cases were selected at random.

[1] A number of other validations also took place, for example an examination of a concept known as fading: where past evidence is forgotten during the learning process at an exponential rate. The fading depends on the decay rate with a long memory able to provide better adaptation, and with a shorter rate giving more dynamic performance, but at the expense of noise. For full details of the validations see [Cockram02]

6.2 Sensitivity analysis

The initial stage of verification of the inspection effectiveness model was to conduct a sensitivity analysis. this involved examining the sensitivity of each individual node of the model on its child nodes and in particular the affect on the top node of the network, which calculates the probability of inspection effectiveness, based on its parent nodes.

The purpose of conducting sensitivity analysis was to determine that the structure of the model is sound, that is that a node affects only its related nodes (parent and child nodes) and no other node within the model. It is also used to show that the initial belief used to initialise the model is consistent with the expert opinion that was elicited. This used a standard Bayesian technique known as sum normal propagation and was applied to the model both after prior-belief values had been applied and after calibration had occurred.

The results of the sensitivity analysis showed that the structure of the basic model was sound and that the most sensitive nodes of the model correspond to the most important attributes found during the elicitation of expert opinion and the least sensitive correspond to the least important attributes.

Furthermore, the results from the sensitivity analysis provides confirmation that the model attributes have the same properties as those factors which Fagan described in his work on software inspections [Fagan86].

6.3 Verification testing

The purpose of the verification testing was to compare the accuracy of the prediction of software inspection effectiveness with the actual effectiveness. A logarithmic rule proposed by Cowell et al. [Cowell RG, Dawid AP et al. 1993] assigns penalty marks on the basis of the equations detailed below:

If $p_m(y)$ is the software inspection effectiveness distribution after mth test case and the actual effectiveness y then the logarithmic score is

$$S_m = -\log_2 p_m(Y)$$

where Y is the predicted probability for the actual value y.

Therefore a correct deterministic prediction will have a value $p_m = 1$ and therefore a score of zero, but a prediction of 0.6 would produce a score of 0.511.

As each test case is applied to the model the total penalty score S which is the sum of the individual penalty scores is

$$S = \sum_{m=1}^{M} S_m$$

If Y has n states, the expectation E_m of the score for the predicted distribution used on the mth test case is given by

$$E_m = -\sum_{k=1}^{n} p_m(y_k)\log_2 p_m(y_k)$$

with a variance V_m of

$$V_m = \sum_{k=1}^{n} p_m(y_k)\log_2^2 p_m(y_k) - E_m^2$$

The standardised test statistic based on the null hypothesis test of significance used Z_M is:

$$Z_m = \frac{\sum_1^m S_i - \sum_1^m E_i}{\sqrt{\sum_1^m V_i}} \,.\, Z_m$$

should tend to a mean of 0 and with a variance of 1 if appropriate predictions are made. The goodness of fit between the models probability forecast and the actual results are assessed using the test statistic Z. If $|Z| < 2$ then the model predictions can be said to be satisfactory. If however $|Z| > 2$ this would indicate a significant mismatch between the model and the data.

Table 1 shows the results from the verification testing of six verifications. Model 1 was the basic model, Model 2 used a fading value of 1, model 3 used a fading value of 0.99, model 4 used half the calibration set (700 inspections), model 5 used twice the calibration set and model 6 used 5 times the calibration set.

The main conclusion to be drawn is that all the models have produced results that are statistically significant in modelling the effectiveness of software inspections. One of the gratifying aspects of the work reported here is that the model (initialised just with the prior belief) gave results using the control data sets that were shown to be a statistically significant improvement over the null hypothesis only after the cumulative evidence of 50 test cases.

6.4 Comparison testing

A comparison was made with a conventional logistic regression model. The results for this model showed that the regression model was a worse predictor in that of the 100 test cases examined 10 fell outside the $Z<=|2|$ criteria, where only 3 fell outside the for the Bayesian Belief model.

7 Applications

There are a number of applications for the work reported in this paper. First, by applying the model of software inspection effectiveness before the inspection takes place, project managers are able to make better use of the inspection resource available.

Second, the model using data collected during the inspection will help in estimation of residual errors in a product. Decisions can then be made if further investigations are required to identify errors close to point of introduction before these are carried into later stages of development and test. This work can therefore be said to have made a contribution to software productivity.

Third the model can be used to determine the effect of varying one or more of the model parameters. For example in a follow-on activity, data obtained from the project was made available to the project manager who used it to improve the planing of subsequent software inspections of a new build of software requirements after a change of user requirements. For this the project manager used the model to determine the effect of using less experienced inspection personnel.

Fourth, software inspection techniques can be applied to inspect safety properties within software, using the methods described within this paper. In this case the inspection techniques are focused on the achievement of safety requirements to address the hazards identified. They can also be applied by independent safety auditors for programmable electronic devices and software working to standards such as Defence Standard 00-55.

The work has also been used to judge the effectiveness of Independent Safety Audits. The use of the model of software inspection effectiveness was tailored to suit the process used and then used to estimate effectiveness of the audit giving an indication of uncertainty in the process and potentially the number of residual errors in the part of the project subjected to audit. This data was then used by the project manager to provide a measure against the whole project by scaling the figures for the portion of the project subject to audit. The data was also used by the independent auditor to provide input to a process improvement activity to improve the effectiveness of independent audits by using the model as a “What If?” tool.

8 Conclusions

Although the work presented here is limited in that only a single, albeit very large, project was examined. It does provide solid evidence that Bayesian models can provide an effective models for inspection effectiveness. we hope that the results of this paper will encourage others to apply our work elsewhere.

References

[Bush90] Bush M Improving software quality; the use of formal inspections at the Jet Propulsion Laboratory. *Proc. 12th International Conference on Software Engineering*, Nice France. 1990.

[Chris88] Christenson D A and Huang S T . Code inspection model for software quality management and prediction. *Proc IEEE Global Telecommunications Conference*, IEEE Computer Soc Press. 1988.

[Christ90] Christenson D A, Huang S T, et al. Statistical quality control applied to code inspections. *IEEE Journal on Selected Areas in Communication* 8(2): 196-200. 1990.

[Cockram02] Cockram, T.G. Validating a Bayesian inspection Model, Open Unviersity technical report 02/13. 2002

[Cowell93] Cowell R G, Dawid A P, et al. Sequential model criticism in probabilistic expert systems. *IEEE Trans. Pattern Analysis and Machine Intelligence* 15(3): 209-19. 1993

[Doolan92] Doolan E P Experience with Fagan's inspection model. *Software Practice and Experience* 22(2): 173-182. 1992.

[Fagan76] Fagan M, Design and code inspections to reduce errors in program development. *IBM Systems Journal* 15(3): 182-211. 1976.

[Hennell89] Hennel M A and Hedley D. The role of static analysis in the validation of safety critical software. *Proc. Software Tools 89*. 1989.

[Herskovits97] Herskovits E H and Dagher A P Application of Bayesian Networks to Health Care. Noetic Systems Incorporated Report NSI-TR-1997-02. 1997.

[Horvitz98] Horvitz E, Breese J, et al. The Lumeriere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *Proc. 14th Conference on Uncertainty in Artificial Intelligence*, Madison WI, Morgan Kaufmann. 1998

[Hugin98] Hugin Expert A/S (1998) Hugin Professional version 5.2 http://www.hugin.dk

[IEEE88] IEEE Standard for software reviews and audits. Institution of Electrical and Electronic Engineers, IEEE Std. 1028. 1988.

[Kelly92] Kelly J C, Sherif J S, et al. An analysis of defect densities found during software inspections. *Journal of Systems and Software* 17(2): 111-117. 1992.

[Knight91] Knight C and Meyers E A. Phased inspections and their implementation. *ACM Sigsoft Software Engineering Notes* 16(3): 29-35. 1991

[Lutz93] Lutz R. Analyzing software requirements errors in safety-critical embedded systems. *IEEE International Symposium on Requirements Engineering, San Diego,* IEEE Comp Soc Press. 1993.

[McCabe76] McCabe T J. “A Complexity Measure.” *IEEE Trans Soft Eng* SE-2(4): 308-320. 1976

[Morgan98] Morgan M G and Henrion M Analytica: A software tool for uncertainty analysis and model communication. In *Uncertainty: A guide to dealing with Uncertainty in Quantitative Risk and Policy Analysis*. New York, Cambridge University Press. 1998.

[OHagan94] O'Hagan A. *Kendall's Advanced Theory of Statistics*. London, Edward Arnold. 1994.

[Pearl88] Pearl J. Probabilistic reasoning in intelligent systems: Networks of Plausible Inference, Morgan Kaufmann. 1988.

[Porter88] Porter A A, Siy H, et al. Understanding the source of variations in software inspections. *ACM Trans on Software Engineering and Methodology* 7(1): 41-79. 1988.

[Porter94] Porter A A and Votta L G. An experiment to assess different defect detection methods for software requirements inspections. *Proc 16th International conference on software engineering*. 1994.

[Reed93] Reed D A. Treatment of uncertainty in structural damage assessment. *Reliability Engineering & System Safety* 39: 55-64. 1993.

[Reeve91] Reeve J T Applying the Fagan inspection technique. *Quality Forum* 17(1): 40-47. 1991.

[Shaw91] Shaw M. Use of Bayes' Theorem and Beta Distribution for reliability estimation purposes. *Reliability Engineering and Systems Safety* 31: 145-153. 1991.

[Votta93] Votta LG Does every inspection need a meeting? Proc. ACM Sigsoft'93 Symposium on Foundations of Software Engineering. 1993

[Wright21] Wright S. Correlation and causation. *Journal of Agricultural Research* 20(7): 557-585. 1921.

[Wright34] Wright S. A method of path coefficients. *Annals of Mathematical Statistics* 5: 161-215. 1934.

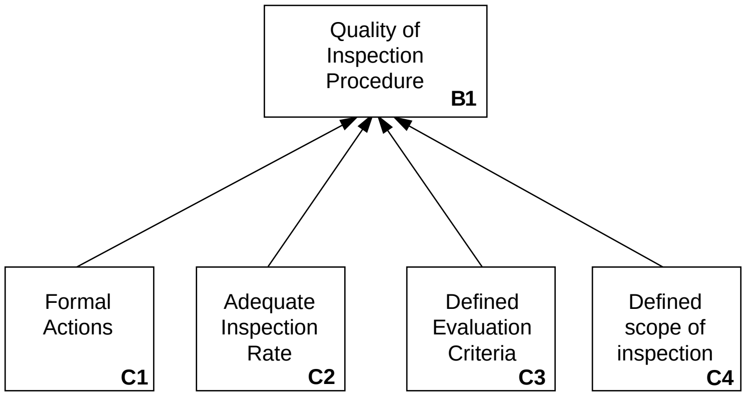
Quality of Inspection Procedure
B1
Formal Actions
C1
Adequate Inspection Rate
C2
Defined Evaluation Criteria
C3
Defined scope of inspection
C4

Figure 1

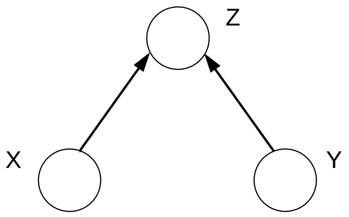

Figure 2

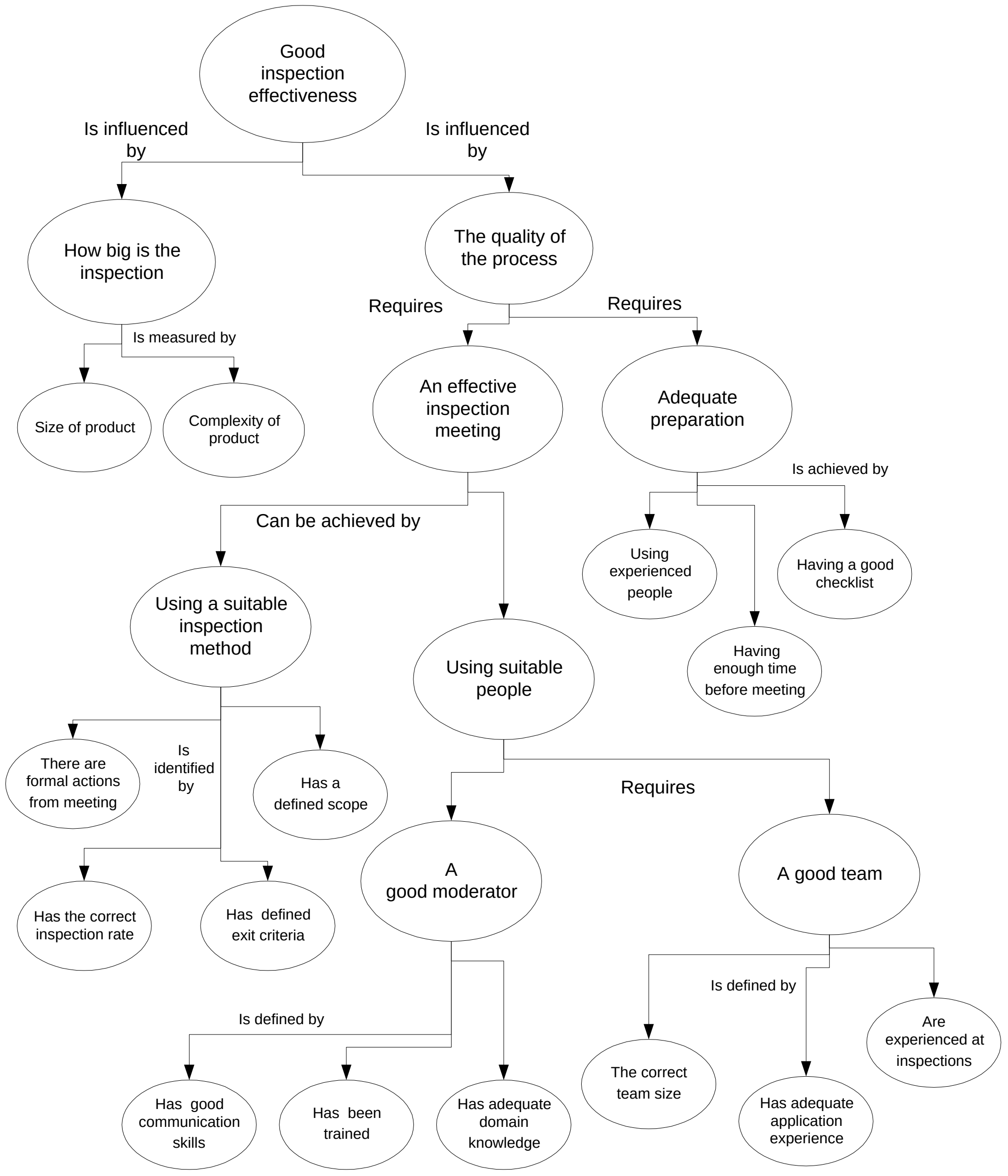
Good inspection effectiveness
Is influenced by
Is influenced by
How big is the inspection
The quality of the process
Is measured by
Size of product
Complexity of product
Requires
Requires
An effective inspection meeting
Adequate preparation
Is achieved by
Using experienced people
Having a good checklist
Having enough time before meeting
Can be achieved by
Using a suitable inspection method
Using suitable people
There are formal actions from meeting
Is identified by
Has a defined scope
Has the correct inspection rate
Has defined exit criteria
Requires
A good moderator
A good team
Is defined by
Has good communication skills
Has been trained
Has adequate domain knowledge
Is defined by
The correct team size
Has adequate application experience
Are experienced at inspections

Figure 3

Model	ΣS_M Log Score	ΣZ_m Significance Test
Model 1 Basic model	76.58	-1.77
Model 2 Calibration Adapted F=1	77.33	-1.76
Model 3 Calibration Adapted F= 0.99	75.93	-1.85
Model 4 0.5 * Calibration	75.74	-1.89
Model 5 2 * Calibration	75.8	-1.88
Model 6 5 * Calibration	75.6	-1.75

Table 1