International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:11, 2007

# Bayesian Belief Networks for Test Driven Development

Vijayalakshmy Periaswamy S., and Kevin McDaid

***Abstract*****—**Testing accounts for the major percentage of technical contribution in the software development process. Typically, it consumes more than 50 percent of the total cost of developing a piece of software. The selection of software tests is a very important activity within this process to ensure the software reliability requirements are met. Generally tests are run to achieve maximum coverage of the software code and very little attention is given to the achieved reliability of the software. Using an existing methodology, this paper describes how to use Bayesian Belief Networks (BBNs) to select unit tests based on their contribution to the reliability of the module under consideration. In particular the work examines how the approach can enhance test-first development by assessing the quality of test suites resulting from this development methodology and providing insight into additional tests that can significantly reduce the achieved reliability. In this way the method can produce an optimal selection of inputs and the order in which the tests are executed to maximize the software reliability. To illustrate this approach, a belief network is constructed for a modern software system incorporating the expert opinion, expressed through probabilities of the relative quality of the elements of the software, and the potential effectiveness of the software tests. The steps involved in constructing the Bayesian Network are explained as is a method to allow for the test suite resulting from test-driven development.

***Keywords*****—**Software testing, Test Driven Development, Bayesian Belief Networks.

## I. Introduction

RECENT research by Wooff et al [1, 2, 3, 4] introduces the use of Bayesian Belief Networks (BBNs) or Bayesian graphical models to aid software testers, developers and managers in the selection and ordering of software tests. BBN's are a probabilistic framework that can combine the software system structure with the expert opinion to select tests to achieve the required level of reliability at system, subsystem or even module level.

Manuscript received January 25, 2006. (Write the date on which you submitted your paper for review.) This work was supported by DKIT as apart of HETAC strand I project.

Vijaya Periaswamy.S., a master's research student of Department of Computing and Mathematics, Dundalk Institute of Technology, Dundalk, Republic of Ireland (phone: +353 429370563; e-mail: Vijaya.Periaswamy@dkit.ie).

Kevin McDaid is a Lecturer of Department of Computing and Mathematics, Dundalk Institute of Technology, Dundalk, Republic of Ireland. (e-mail: Kevin.McDaid@dkit.ie).

With the assistance of a major international company, Wooff et al have trialed the approach on a number of systems. This research is built on their work but goes beyond existing work by investigating, through a real software project, whether this approach can be applied to a system developed using a test-driven development (TDD) methodology.

The paper is structured as follows. Section II describes the methodology pioneered by Wooff et al to use BBNs to assess the quality of tests based on their contribution to the reliability of the software system. Section III discusses the application of the methodology within the test driven development scenario.

Section IV illustrates our work through a case study that applies the approach to an existing system. W explain how the resulting model may be used to assess the quality of the tests that are generated through TDD and to propose supporting tests that can contribute to a significant reduction in the remaining reliability of the module examined. Section V concludes the work.

This paper presents some early results from a study whose broader objective is to research the application of the BBN methodology to systems developed using test-driven development.

## II. Bayesian Belief Networks

The theory of BBNs, developed over the last two decades, has, through the availability of good computing facilities, become an increasingly powerful tool for the solution of complex decision problems where a large number of factors contributing to overall uncertainty [5].

A BBN is a network of nodes connected by directed links with a probability function attached to each node. The nodes represent uncertain variables and arcs represent the casual relationships between the variables.

The probability tables for each node provide the probabilities of each state of the variable represented by that node. Nodes without parents require marginal probabilities while for the nodes with parents, these are conditional probabilities for each combination of parent state values [6].

In effect, if there is a directed link from one node (parent) to another (child) in a BN then the probability of the child is evaluated conditionally on the values of the node from which the link originates [7]. Sections II and IV describe a particular BBN in more detail.

In general, the network is characterized by providing a formal framework for the combination of data which, in our case, allows test results to be combined with experts' opinion to assess the quality of software tests.

### *A. How to Build BBNs for Software Testing?*

The BBN methodology focuses on major software actions and the probability that there is no input that is processed incorrectly by the action.

A software action is effectively a self contained collection of software code responsible for a piece of processing that can be practically tested. In the following example we examine a single discrete software action. In practice a more complicated system is likely to contain one or more transactions each of which may consist of a series of software actions.

An example would be a transaction in an ATM machine which may consist of a series of actions including checking of card, requesting of information such as PIN and account details, processing of information to complete selected financial request and finally the delivery of the cash and receipt.

Generally, the first key issue in the development of the representative BBN involves the sequencing of these actions linked to an inputted test. This is typically the first stage in the development of a BBN. We now discuss the process in more detail.

### *B. Stages Involved in Structuring BBNs*

The different features involved in creating a BBN to aid testing of a software module are depicted in Figure 1.

Initial development consists of identifying the Software Actions and structuring the BBN based on the possible inputs and the relationships between SAs. Sequencing of SAs is particularly important in the process.

The next stage involves the development of domain nodes for each software action based on partitioning the possible test inputs for the Software Actions. The inputs are the possible test values and the input spaces can vary for different SAs. Construction of the groupings is based on expert beliefs regarding the likelihood that certain test inputted values are more or less likely to fail than others.

Using a well-quoted practical example, the processing of a PIN number may depend on the length of the number and thus require two domain nodes to represent the two subgroups of possible tests, namely long numbers and short numbers. Technically the inputs for any domain node should be exchangeable in the sense that the processing of any input within a group by the associate software action does not give any more information then the processing of another input within the group.

Once the input nodes are established the method inserts nodes to represent the problem that at least one input of each exchangeable type fails.

The structuring activity is completed through the creation of the BBN for the software action. This involves specifying nodes for the possible problems that could cause the software to fail. The node representing the overall quality of the software, through the probability that at least one failure causing fault remains, follows from these problem nodes.

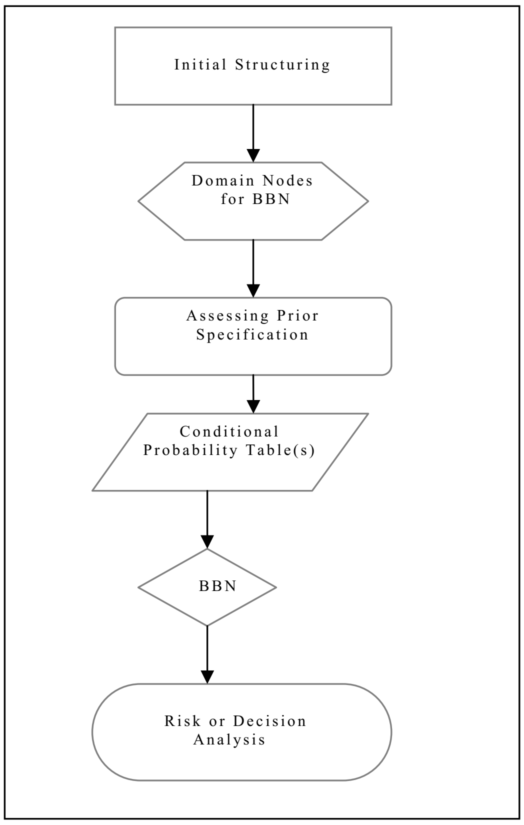
Initial Structuring
Domain Nodes for BBN
Assessing Prior Specification
Conditional Probability Table(s)
BBN
Risk or Decision Analysis

Fig. 1 Structuring a BBN

Once structuring is complete, the next stage is assessing prior specification for all the nodes. The probability values defined for the root nodes in the network are elicited and the conditional probability values for the child nodes are assigned based on the structural relationships between nodes.

When we later take a practical example we shall see that the BBN for a software action can be seen to consist of two separate layers. The first relates to faults and the source of faults in the action. We label this the Fault layer, the second looks at possible inputs and tests and the likelihood that the action may process an input incorrectly. This we term the Test layer.

All the results required in the approach are computational and it can performed using packages like HUGIN$^{TM}$, Netica$^{TM}$, MSBNX etc...The stages given above are an outline for the development of BBN's, for further details we refer to [1].

We demonstrate this approach as part of our ongoing study and describe the overall aspects of the implementation of our approach in the following section We next term our attention to the area of Test Driven Development before we investigate whether the BBN methodology can be applied to code developed using this methodology.

## III. Test-Driven Development (TDD)

TDD is one of the emerging software programming techniques that combine test-first development, where you write the tests before you write the code to fulfill that test and with refactoring of tests [8]. The primary goal is the requirement specification and not validation. To be precise it follows that you design before you write the functional code.

TDD does not replace traditional testing. Instead it defines a way to ensure effective unit testing. The outcome is that the resulting tests are the working examples for the invoking code, thereby satisfying the specification of the code. The question in this paper is how these tests should be incorporated within the BBN methodology proposed.

TDD has been found to be a much more flexible approach to the client's ever changing requirements by embracing iterative development techniques. This will cause a natural and longer testing period and improve the quality of the application.

TDD, when implemented, results in a suite of automated tests. Automated tests have significant advantages. By maintaining a suite of tests that are both repeatable and automated we can benefit by updating the changes needed for the code design with prior knowledge that any deviations or errors will be traced by re-running the test suite [9].

However, the key question remains regarding the quality of the tests that are generated through TDD. In practice a further suite of unit tests are added to this harness, through refactoring, to ensure the modules are sufficiently reliable.

This paper studies the methodology to establish the reliability of the existing tests and to propose supporting additional tests.

TDD has its origin in Java and Smalltalk [9] and has grown to include a number of supporting tools. These include CUnit, DBUnit, JUnit, OUnit, NUnit and VBUnit. These tools are represented by means of 'xUnit framework' following the language agnostic version of different tools. Based on the application needs, these frameworks can be ported to many different platforms and languages

Generally unit tests are coupled tightly to application code, as they grew up with the code from scratch. If it is the case, one should normally change the unit test to test the functionality of the unit instead of design internals. But if the deadline is tight, as is often the case with software development, there is often only time to fix the unit tests and to ensure they continue to run.

TDD offers clear advantages for the software professional. However, it is not perfect and it is difficult for the professional to establish whether module code has been tested sufficiently. The BBN methodology may overcome this issue.

The next section examines a test suite consisting of a set of unit tests developed for a module of a real system and explores whether the BBN approach can be used to assess the quality of the tests.

## IV. Case Study

We next examine one module from a recently developed software system to demonstrate the methodology. Ongoing research is applying the methods to a large number of modules. The module, named Module X, takes two floating point inputs labeled D and T, and returns an array of floating point values following a set of calculations. This piece of code, which is an important task within a bigger system, is treated as a single software action.

In general software actions may involve a number of modules and thus unit tests, developed through TDD, may combine to indicate the reliability of software actions. This module includes 7 tests.

This system is developed in Java and the TDD framework used is JUnit 3.8.1. The module and the unit tests were run in the Eclipse 3.0.2 platform. Details of the company and the software system under investigation are not revealed for confidentiality reasons.

We present an example of a BBN designed for the module in question following the stages discussed in Section II.

### *A. Construction of BBN*

For this single module there is one Software Action requiring two inputs with 7 tests already in place following test driven development. The question arises as to whether these tests are sufficient to establish the reliability of the system or whether further tests are required. We construct the BBN to answer these questions.

The entire BBN is shown in Figure 2. It centers on the reliability node N which gives the probability that the software module contains no faults. This is the crucial measure of the reliability of the module.

The two floating point inputs to the module are D and T. The first step is identify the classes of inputs that are exchangeable, that is the separate group of inputs within which no one pair of inputted values for D and T is believed to be more likely to fail then another.

After careful examination of the code, in collaboration with the developer, it was decided that the inputs could be separated into five groups of exchangeable inputs. These are listed in the table below together with a description of allowed values. The breakdown can be seen to depend strongly on the product of the two inputs.

Each of the nodes, L, M, H, D0 and T0 feeding into N represent the probability that at least one fault from the exchangeable input group will fail due to a fault that affects these inputs.

To allow the elicitation of expert opinion in a natural way the BBN includes the nodes L*, H*, D0* and T0* to represent the probability that there is a problem in the code which may affect the particular set of inputs alone.

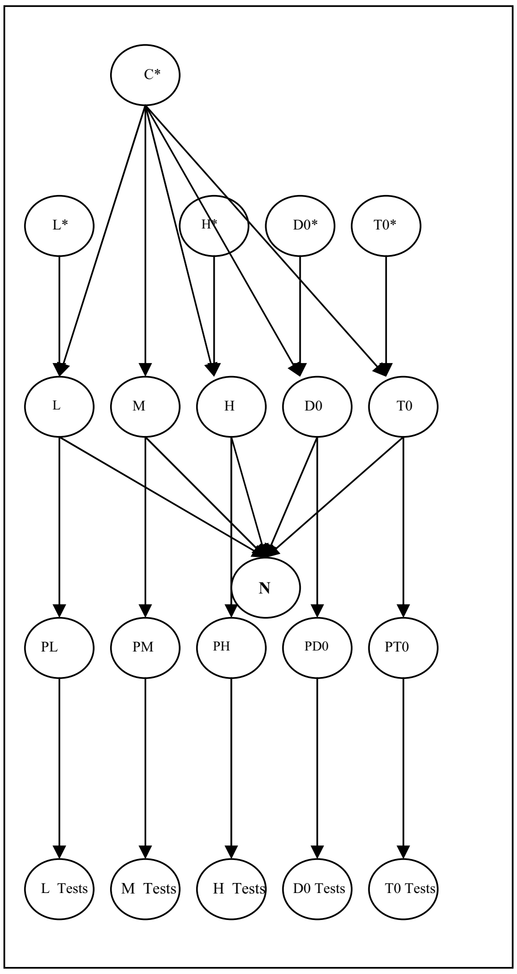

Fig. 2 BBN for the Module X

TABLE I
LIST OF ALLOWED VALUES FOR THE DIFFERENT DOMAIN NODES

| S.No | Group Labels | Description |
|---|---|---|
| *1* | L | D > 0 & T > 0 & D,T Low |
| *2* | M | D > 0 & T > 0 & D,T Medium |
| *3* | H | D > 0 & T > 0 & D,T High |
| *4* | D0 | D = 0 & T > = 0 |
| 5 | T0 | D > 0 & T = 0 |

Thus, for example, L* is the event that there is a problem in the code which produce faults for inputs of the exchangeable group L but not for inputs of any of the other groups M, H, D0, T0. The C* node reflects the chance that there is a problem which may affect any input. Note that there is no M* node as it is felt that the any problem which affected this node would affect any of the others and thus could be replaced by C*.

The lower half of the BBN deals with the possible test inputs. In particular the nodes PL, PM, PH, PD0 and PT0 give the proportion of inputs of each exchangeable group that are likely to fail conditional on the state of the parent nodes. From these we have the test nodes L tests, M tests, H tests, D0 tests and T0 tests which represent the possible tests.

## *B. Quantify Beliefs*

The model can be quantified through an elicitation process. For this BBN we start specifying probabilities for the root nodes based on the Expert's judgment. The conditional probability tables for the nodes L, M, H, D0, T0 and N are often deterministic in structure and require no elicitation activity. The proportion nodes PL, PM, PH, PD0 and PT0 however require an amount of expert input. In particular the expert must assess the likelihood that all inputs will fail if there is a problem with at least one input. Furthermore the expert must provide indications as to the best beta distribution to model the likely proportion from 0 to 1 that will fail given a problem that affects the particular group of inputs. The final group of nodes representing the probability of test success and failure follow from the proportion nodes.

## *C. Test Procedure*

Testing of the module involves the selection of pairs of D and T values with each pair associated with a one of the exchangeable groups. Each test will add one test node to the appropriate proportion node in the BBN. Note that when running the tests all other inputs should be selected randomly.

If the test passes in practice then this information is added to the BBN through the instantiation of the appropriate test node. This then reduces the chance of failure of similar tests. Most importantly the success of a test should increase the reliability of the module and reduce the chance of failure of all tests as they are indirectly linked through the C* node. If a test fails then the BBN can indicate where the problem is most likely to lie. Once the software is repaired a redesign of the BBN may be necessary.

This approach provides a probabilistic assessment of the reliability of the software being tested before and after the testing process and helps to choose the test suite which maximizes the conditional probability of software acceptability.

The tests can be chosen by searching all possible combinations to maximize the achieved reliability though node N.

The approach can be applied when tests are already in place though TDD. In our case study, we have taken initially seven possible test cases within the test suite and analyzed the reliability outcome in the node N. The presentation of our results and brief discussion concludes the work.

## V. Results and Discussion

Following construction of BBN and elicitation of all expert opinion but before inclusion of test results it was found that the reliability of the module was 64.33 %.The results are shown in Table 2.

To increase this reliability the BBN methodology proposes that the tests should be selected based on their contribution, assuming they pass, to this reliability. It was found that a test using inputs from group H (labeled H Test 1) would increase the reliability by 21.03% which was better than the rest of all other tests. Hence H Test1 should be carried out first. This is then considered to have passed and the appropriate test node instantiated.

The process again examines the five possible distinct test cases next to apply and chooses based on their contribution to the achieved reliability. If we continue on this tack, wishing to achieve over 99.50% reliability say, then Table 2 shows the order in which tests would be selected.

That results in an order for tests as follows D0 Test1, L Test1 followed again by H Test2, L Test2, H Test3, L Test3, T Test1, L Test4, H Test4, L Test5, L Test6, H Test5, D0 Test 2, and H Test5 subsequently increasing the reliability by 6.58, 3.37, 2.08, 0.62, 0.53, 0.31, 0.19, 0.18, 0.17, 0.11, 0.07, 0.06 and 0.05 percentages at each successive test runs.

As the table shows 11 tests are required to achieve the desired reliability of 99.50%. The eleven tests are broken down as four of type H and five of type L and one of type D0 and one type of T test with no tests of type M necessary.

It is clear to see the diminishing effect on reliability as further tests are added. The point at which further testing is no longer economically viable is a decision for each developing organization. Note that, these are scenarios where further tests are completely redundant. This will occur where the expert opinion indicates that either all tests pass or all tests fail.

The selection of tests described hitherto does not account for the Test Driven Development methodology. Under this approach a number of tests are in place prior to any subsystem testing. These must be allowed for in both the initial specification of the reliability and in the selection of further tests.

The existing test suite consists of seven unit tests to verify the functional aspects of the module. These tests were matched to the exchangeable input groups used in the specification of the BBN. On examination it was found that the test suite consists of three H Tests, three L tests and one D0 Test. The test sets were not run with a specific order in mind. Appropriate test nodes were added to the BBN and instantiated to reflect this information.

TABLE II
List of Allowed Values for the Different Domain Nodes

| Test | Test Name | Reliability Attained (%) |
|---|---|---|
| None | | 64.33 |
| 1st run | H Test 1 | 85.36 |
| 2nd run | D0 Test 1 | 91.94 |
| 3rd run | L Test 1 | 95.31 |
| 4th run | H Test 2 | 97.39 |
| 5th run | L Test 2 | 98.01 |
| 6th run | H Test 3 | 98.54 |
| 7th run | L Test3 | 98.85 |
| 8th run | T Test 1 | 99.04 |
| 9th run | L Test 4 | 99.22 |
| 10th run | H Test 4 | 99.39 |
| 11th run | L Test 5 | 99.50 |
| 12th run | L Test 6 | 99.57 |
| 13th run | H Test 5 | 99.63 |
| 14th run | D0 Test 2 | 99.68 |
| 15th run | H Test 6 | 99.70 |

It is interesting to note that, as with the derived test cases, these automated tests do not include inputs of type M or T0

The seven TDD tests increase the reliability by 34.52% from 64.33% to 98.85%. Using the methodology described we determined that to obtain reliability of 99.50% we must apply one further tests of type H, two test of type L and one test of type T.

## VI. Conclusion

This paper illustrates an emerging technique to aid the selection of software tests. The BBN methodology is both a flexible and powerful way to organize the belief modeling to support the key activities of software testers and managers.

We apply the methodology through a case study of a single module developed using test driven development. By its nature, this approach yields working tests prior to extensive subsystem and system testing. We show that these can be included in the methodology to guide the selection of further tests.

In future research we intend to apply the methodology to a larger system.

## Acknowledgment

The authors acknowledge the support of the TSR fund administered by the Council of Directors of Institutes of Technology in Ireland as well as the research secondment programme at Dundalk Institute of Technology.

## REFERENCES

[1] David A. Wooff, Michael Goldstein, and Frank P.A. Coolen, "Bayesian Graphical Models for Software Testing", *IEEE Transactions Software Engineering,* May2002, pp 510-525.

[2] Kearton Rees, Frank Coolen, Michael Goldstein, and David Wooff, "Managing the Uncertainties of software testing: a Bayesian Approach". *Quality and Reliability Engineering International,* 17, in 2002 pp 191-203.

[3] F.P. Coolen, M. Goldstein, and D.A. Wooff, "Using Bayesian statistics to support testing of software systems". *Proceedings of the 16th Advances in Reliability Technology Symposium,* in 2005, pp 109-121.

[4] F. P. A. Coolen and M. Goldstein and D. A. Wooff, "Project viability assessment for support of software testing via Bayesian graphical modeling", *In: Safety and Reliability, Lisse: Swets & Zeitlinger*, in 2003, pp 417-422.

[5] Jensen, F.V.,".*Bayesian Networks and Decision Graphs".* New York: Springer 2001

[6] "Bayesian Belief Networks", Available: http://*www.hugin//developer.com//*

[7] Norman E. Fenton, Martin Neil, "Software Metrics: roadmap", *Proceedings of the Conference on the future of Software Engineering*, ACM Press, 2000, pp 357-370.

[8] "Introduction to Test-Driven Development (TDD)", *Available: http://www.agiledata.org//*

[9] "Improving Application Quality Using Test-Driven Development (TDD)", Available: *http://www.methodsandtools.com//*

**Vijaya Periaswamy S.**, received her Engineering Degree at REC, Trichy, India and after three years in Software Development and Training, Vijaya now researches the application of Bayesian Belief Networks to Software Testing under the guidance of Dr. Kevin McDaid as part of her Msc.

**Kevin McDaid** received his BSc, joint honors, in Mathematics and Mathematical Physics and his Msc in Statistics from University College Dublin. He received his PhD in Statistics from Trinity College Dublin. He currently lectures in the Computing Department at Dundalk Institute of Technology where he specializes in teaching Computer Mathematics and Programming. Kevin is interested in the application of statistical techniques to Software Engineering. His research is currently focused on decision problems in software development and he has conducted significant research into the problem of when to stop testing software.