Cost-benefit analysis of software testing with a causal model

MSc Thesis Proposal

Niek Scheers

May 6, 2009

Radboud University Nijmegen, Computing Science
Student number: 0539627
Supervisor: dr. P.J.F. Lucas
External Supervisor: Info Support B.V.
Sander Mak
Erno de Weerd
Contents

1 Introduction ........................................ 3

2 Problem statement ................................ 4
   2.1 Research questions .......................... 4

3 Literature study .................................. 5
   3.1 Critique on metric activities .............. 5
   3.2 Bayesian Artificial Intelligence ............ 6
      3.2.1 Bayes’ Theorem ....................... 6
      3.2.2 Bayesian Networks .................... 7
      3.2.3 Causal reasoning ....................... 9
      3.2.4 Decision networks ....................... 10
      3.2.5 Knowledge engineering with Bayesian Networks ...................... 10
   3.3 Causal prediction models in literature ...... 13
      3.3.1 Predicting software defects in varying development lifecycles .......... 13
      3.3.2 Is my software “good enough” to release? ...................... 13
   3.4 Software reliability modelling .............. 14

4 Research approach ................................ 18

5 Research objective ................................ 19

6 Planning ............................................ 19

References ......................................... 19
1 Introduction

One of the most important, if not the most important, thing in software engineering is software testing. Testing is an important part of quality assurance. Testing reviews the specification, design and implementation of software. The associated costs of failing software are the biggest motivation for thorough testing. Software development organizations spend between 30% and 40% percent of their total project effort on testing [Pressman, 2001]. An efficient testing process and effective testing can save a company money. The effectiveness of testing can be indicated by the number of found defects divided by the software size while the efficiency of the process can be determined by the number of found defects divided by the effort put into testing. These are very coarse indicators, but the point is that measurement of the software engineering process is necessary to provide insight into it. This insight is needed to improve the software process. Without measurement, judgement can only be based on subjective evaluation. With measurement, judgement can be based on objective facts and possible trends can be identified. This makes better judgements possible which ideally improve the software process.

Measuring some property of software is done with software metrics. The IEEE Standard Glossary of Software Engineering Terms defines a metric as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute”. Well known traditional metrics are Lines Of Code and Function Point analysis. Both metrics indicate the size of a software system. More advanced metrics indicate for example the expected size of a software component. These metrics use prediction models as measure. Prediction models include mostly other metrics to predict properties of the software system or engineering process. The goal of most of the metrics available can be characterized into two activities [Fenton and Neil, 2000]:

- The desire to assess or predict effort/cost of development processes
- The desire to assess or predict quality of software

Cost-benefit analysis - The process of weighing the total expected costs against the total expected benefits of one or more actions in order to choose the best or most profitable option - can be used in assessing risks and how to manage them. Metrics make risk assessment and cost-benefit analysis possible because they supply measures for the software engineering process. But lately, a critique is given on metrics for risk assessment. Fenton and Neil [Fenton and Neil, 2000] state that “traditional metrics approaches, often driven by regression-based models for cost estimation and defects prediction, provide little support for managers wishing to use measurement to analyse and minimize risk”. Especially the analyzing part is not provided by most traditional metric approaches. For true risk assessment a metric needs to provide:

- Quantified decision support, so decisions can be backed by concrete figures.
- An explanatory model, so that measures are not misinterpreted and false derivations are not made. An example of misinterpretation is the “Goldilocks Conjecture” [Fenton and Neil, 1999a]. This research showed that there is an optimal size for software modules based on found defects. Larger modules were better than smaller modules, because less defects were found in the larger modules while testing. This undermined the whole divide and conquer engineering style. The real reason for finding less defects was that larger modules were a lot harder to test. Therefore, less defects showed up during testing and the wrong conclusion was made that larger modules are better. This is a problem of not taking into account all of the underlying causal relations found in the software engineering process.
- The ability to deal with uncertainty, because uncertainty is inherent in the software engineering process. “Software development is a complex human enterprise carried out in problem domains and under circumstances that are often uncertain, vague, or otherwise incomplete. Development must progress, however in the presence of those uncertainties” [Ziv et al., 1996] [Ziv and Richardson, 1997a] [Ziv and Richardson, 1997b]. Therefore a prediction model should be able to take into account uncertainty in its input and output.

These requirements ask for prediction models that can handle [Fenton and Neil, 2000]:
The solution introduced is the use of causal models as prediction model. Bayesian Networks are such causal models and lately popular in research to provide prediction models to assist in true risk assessment.

2 Problem statement

A Software Factory is a development environment configured to support the rapid development of a specific type of application. It is a collection of software engineering project experience combined into guidelines and processes stating best-practices. Info Support, the company where this research takes place, use their own software factory, called Endeavour, to control software engineering processes. Using Endeavour, Info Support aims to ensure software engineering process predictability and software quality.

Endeavour uses metrics to measure different aspects of the software engineering process. These metrics give Info Support a certain amount of insight into a software engineering process. But besides the metric data being collected, there is no support that provides insight specified towards software testing. Info Support would like to have more insight into testing, so it can perform a cost-benefit analysis with this information. The analysis can be used for risk assessment and to steer the software testing. Info Support is for example interested in how a change in the effort put into testing influences (parts of) the software engineering process and the resulting software product. A model is needed that can predict the testing process, outcome and other aspects that influence or originate from software testing.

In this paragraph I will describe some aspects which, according to other research, need to be taken into account in creating a prediction model. Creating prediction models for true risk assessment has been directed towards the incorporation of diverse process and product variables. The reason behind this is to explain all cause and effect relations present in the researched domain. But are causal models that incorporate more domain information also better for risk assessment? It is logical to think that more domain information can explain the cause-effect relations better, but will the prediction with such a model also be better? The use of expert judgement is also argued to be an helpful extra source of input for a prediction model. Will expert judgement truly help in the prediction? Human participation is seen as a source of uncertainty in software engineering [Ziv et al., 1996]. Will this introduction of an extra uncertainty help in making a correct prediction? It is also argued that it is wise to incorporate and model uncertainty in a prediction model since the input information comes from a software engineering process where uncertainty is inherent. Will the explicit notion of uncertainty in the prediction model help in making the prediction more understandable? Providing an explanatory prediction model is an important point for true risk assessment. If this is missing, misinterpretation of risk is possible.

2.1 Research questions

The following main research question is stated for the problem statement:

How to create a prediction model to support cost-benefit analyses of testing in the software engineering process?

The software engineering process refers to the software engineering process controlled by the software factory Endeavour. The prediction model will use existing empirical information available in Endeavour. Hereby, the model can be used without requiring new information to be collected in existing Endeavour projects.
Subquestions

1. What are the testing strategies used in Endeavour?
2. What data is collected in Endeavour that relates to testing?
3. Which variables can be identified that give information on testing?
4. What is affected in the software engineering process by changes in testing?
   (a) And what are the associated relationships?
5. How can the variables and relations be modelled in a model?
   (a) What will the possible values of the variables be?
   (b) Which model is a good probability estimator?
   (c) Will a good probability estimator reflect causality and can it be used as a explanatory framework?

3 Literature study

3.1 Critique on metric activities

One of the major objectives in metrics research is to produce metrics which are good and early predictors of software quality. Usually this work has involved regression-based models of the form \( \text{effort} | \text{quality} = f(\text{size}) \). Fenton and Neil discuss \cite{Fenton and Neil, 1999a, Fenton and Neil, 1999b, Fenton and Neil, 1999c, Fenton and Neil, 2000, Fenton, 2006} that existing models are incapable of predicting defects accurately using size and complexity metrics alone. The straightforward relationship assumption between metrics and prediction is often wrong. Size and complexity metrics can’t accurately indicate defects, because defects are not caused by the software size or the software complexity. Study shows there is correlation, but this is not a entirely causal. Important to notice is that “it is wrong to mistake correlation for causation. An analogy would be the significant positive correlation between IQ and height in children. It would be dangerous to predict IQ from height because height doesn’t cause high IQ”. This is a misunderstanding of the notion of cause and effect and of not incorporating all causes into the prediction model to explain an effect.

Metrics are supposed to provide information to support quantitative managerial decision-making during the software life-cycle. Fenton and Neil propose the use of causal models for true decision support potential. Good support for decision-making implies support for risk assessment. Unfortunately, most metrics approaches do not provide such support and fail in giving true decision support. Causal models can provide an explanatory structure to explain events that can then be quantified. This is missing in traditional models and makes real quantified risk assessment possible. A another advantage of causal models is that they allow all the evidence to be taken into account, even when different evidence conflicts \cite{Neil and Fenton, 2005}.

Metric models tend to focus too much on product metrics. Hereby they ignore an important input source that could lead to better predictions. “Again and again experience dictates that it is good managers and designers that determine the difference between failure and success” \cite{Neil and Fenton, 1996}. So why not use this key variable as input in a metric activity?

Fenton et al. \cite{Fenton et al., 2007} propose several prediction models based on this work, focusing on defect and resource modelling/prediction. The intention of these models is to be a decision support tool for software project managers. Size and complexity metrics are combined with input from managers about the quality of the software process to predict residual defects. The residual defects measure is seen as the quality of software.

Because uncertainty is inherent in the software development process, a prediction model needs to know how to cope with uncertainty. To prevent misinterpretation about an uncertain correlation between artifacts it should clearly model cause and effect relationships including the

The critique on traditional metric activities and the requirement for handling uncertainty result in a list of requirements for metric activities. Metric activities that want to enable true risk assessment should incorporate [Fenton and Neil, 2000]:

- diverse process and product variables
- empirical evidence and expert judgement
- genuine cause and effect relationships
- uncertainty
- incomplete information

The metric activity should also provide the following to enable true risk assessment:

- Quantified decision support
- An explanatory model

The above researches propose Bayesian Networks as the technique for causal prediction models that support true risk assessment.

3.2 Bayesian Artificial Intelligence

Bayesian Networks are a computer technology for dealing with probabilities in Artificial Intelligence. A Bayesian Network (BN) is a graphical structure representing an uncertain domain and allows reasoning about that domain. The theory behind BNs is based on the Bayesian philosophy; “in order to understand human opinion as it ought to be, constrained by ignorance and uncertainty, the probability calculus is the single most important tool for representing appropriate strengths of belief” [Korb and Nicholson, 2003].

3.2.1 Bayes’ Theorem

The origin of the Bayesian philosophy lies in a theorem of the probability calculus, called the Bayes’ theorem:

**Theorem 1** (Bayes’ Theorem).

\[ P(h|e) = \frac{P(e|h)P(h)}{P(e)} \]

It asserts that the probability of hypothesis \( h \) conditioned on some evidence \( e \) is equal to its likelihood \( P(e|h) \) times its probability prior to any evidence \( P(h) \), normalized by dividing by \( P(e) \) (so that the conditional probabilities of all hypotheses sum to 1). Controversial in the Bayesian philosophy is that this is a right and proper way of adjusting our beliefs in our hypotheses given new evidence. This is called conditionalization: After applying Bayes’ theorem to obtain \( P(e|h) \), one adopts that as one’s posterior degree of belief in \( h \), or \( \text{Believe}(h) = P(h|e) \). Conditionalization advocates belief updating via probabilities conditional upon the available evidence. It identifies posterior probability (the probability function after incorporating the evidence) with conditional probability (the prior probability function conditional upon the evidence).

Let’s see the Bayes’ theorem in action. Suppose we want to know what the probability is that a woman has breast cancer, given the fact that she has tested positive for a cancer screening. The prevalence of breast cancer is 1 in a 100. The cancer screening has a false positive rate of 0.2 (20% of the woman without cancer test positive) and that is has a false negative rate of 0.2. A Bayesian Network is also known as Bayesian Belief Network (BBN).
0.1 (10% of the woman with cancer test negative). Because of the fact that 90% of the woman that tested positive have cancer, one might think that the probability that the woman has cancer given the test is high. But if we calculate the probability with Bayes’ theorem we see a different conclusion:

\[
P(\text{Cancer} | \text{Pos}) = \frac{P(\text{Pos} | \text{Cancer}) P(\text{Cancer})}{P(\text{Pos})} = \frac{P(\text{Pos} | \text{Cancer}) P(\text{Cancer})}{P(\text{Pos} | \text{Cancer}) P(\text{Cancer}) + P(\text{Pos} | \neg \text{Cancer}) P(\neg \text{Cancer})} = \frac{0.9 \times 0.01}{0.9 \times 0.01 + 0.2 \times 0.99} = \frac{0.009 + 0.198}{0.009 + 0.198} \approx 0.043
\]

This is because people tend to neglect the base rate. The change to have cancer is only 0.01. No figures were given on the number of women that participate but do not have breast cancer.

[Korb and Nicholson, 2003]

### 3.2.2 Bayesian Networks

Bayesian Networks are the computer technology to reason about an uncertain domain. They are suitable for: 1) Problem modelling objective, e.g. for pure numeric prediction without a need to explain results a “black box” model such as a neural network can be sufficient; 2) Sufficient knowledge about the problem domain is available; 3) Complexity of the problem is decomposable.

A BN, \( B = (G; Pr) \), is a graphical structure, \( G = (V(G); A(G)) \), where nodes, \( V(G) = \{V_1, ..., V_n\} \), represent variables about a domain and directed arcs, \( A(G) \subseteq V(G) \times V(G) \), represent direct connections between them. These connections are often causal relationships. The strength of the connection between variables is quantified by conditional probability distributions associated with each node. A constraint in the network is that there can’t by any directed cycles. Such networks are known as Directed Acyclic Graphs. The structure of a BN is called the qualitative part; it captures the relationships between variables. The quantification of the relationships is called the quantitative part; this consists of the joint probability distribution, defined in terms of conditional probability tables (one for each variable) according to the structure of the graph.

The fundamental assumption is that there is a useful underlying structure in the domain modelled as a BN, and that because of this not all nodes are connected to each other. The structure of a BN implies that a value of a node is conditional only on the values of its parents. This allows the joint probability distribution be represented in a computationally tractable way:

\[
Pr(V_1, ..., V_n) = \prod_{i=1}^{n} Pr(V_i | Parents(V_i))
\]

Here \( V_i \) denotes a random variable associated with an identically named node, and \( Parents(V_i) \) denotes the parents of that node. Because of the structure of a BN, the amount of probabilistic information that must be specified, exponential in the number of variables in general when ignoring the independencies represented in the graph, is greatly reduced. [Lucas, 2000, Korb and Nicholson, 2003]

An example of a BN, that models a simplification of the relevant variables in the diagnosis of two causes of fever, is given in Figure [Lucas, 2000]. An arc resembles a direct causal relation or some other influence. Absence of an arc between variables means that they do not influence each other. It is known that flu causes myalgia (muscle pain) and fever. Fever causes body temperature to change. Pneumonia is another cause of fever besides flu. In this figure the associated joint probability distribution is given and the Conditional Probability Table(CPT) \( Pr(FE | FL, PN) \) has been assessed with respect to all possible values of the variables \( FE, FL \) and \( PN \).
Reasoning with BNs is done by updating beliefs given some evidence. Updating beliefs is done by computing the marginal probability distribution for every variable in the network. This is called probabilistic inference. Different algorithms, exact and approximate, exists for inference. A well known algorithm is from Pearl [Pearl, 1988]. Most algorithms repeatedly apply Bayes’ theorem and use the conditional independencies encoded in the network structure. A description of algorithms is beyond the scope of this review, as most popular algorithms are implemented in BN modelling software and are ready to use. The prior marginal probability distribution for every variable in the fever diagnostic network is shown in Figure 2. Updated posterior marginal probability distributions is shown in Figure 3 after evidence about TEMP is entered.

Figure 1: BN with associated joint probability distribution $Pr(\text{only probabilities } Pr(X = y \mid \text{Parents}(X)) \text{ are shown, as } Pr(X = n \mid \text{Parents}(X)) = 1 - Pr(X = y \mid \text{Parents}(X)))$.

For large BNs assessing all probabilities can become infeasible. Special forms of BNs can be
used to solve this problem. To have a simpler structure one can use Naive (independent) BNs and Tree-Augmented BNs. To have a simpler assessment of the conditional probabilities one can use causal independence BNs.

### 3.2.3 Causal reasoning

There are two different opinions in the community of BN researchers about the proper understanding of the relation between causality and BNs [Korb and Nicholson, 2003]. The majority opinion is that there is no reason for causal interpretation of a BN. Arcs in a BN model a probabilistic dependency. This can also be a causal dependency, but it doesn’t have to be; variables in a model can be re-ordered so that arcs are turned around. Knowing this, both models cannot be causal. If a BN represents a real world probability distribution then it is assumed that some causal model is its source. Causal models are important because inferences can be done in a different way. Figure 4 shows a BN that represents a causal structure [Pearl, 1988]. It models an alarm that detects burglary, but also responds to minor earthquakes. Two neighbours, John and Mary, promise to call the police when they hear the alarm. John sometimes confuses the alarm with the phone ringing. Mary likes loud music and sometimes doesn’t hear the alarm. Given the evidence about who has and hasn’t called, you’d like to estimate the probability of a burglary. Because this is a BN we know that if JohnCalls is true, this will also raise the probability of MaryCalls being true. But if we intervene on this and force John to call, this inference is no longer valid. The variables JohnCalls and MaryCalls provide information about each other through their common cause Alarm. When forcing John to call, we don’t know anything about whether Mary will call or not. The belief propagation from JohnCalls, to Alarm, to MaryCalls is now wrong. This is called causal intervention [Korb and Nicholson, 2003]. Causal intervention can be modelled in two ways. The first one is to simply cut the arc from Alarm to JohnCalls. Forcing John to call tells us nothing about the alarm, earthquakes and burglaries - which is correct. This Pearl’s cut method. The second method is creating an augmented model by adding a new parent to JohnCalls that provides the probabilistic impact on the intervention on John calling. Propagation with this new node, when it is observed, will provide correct causal inference. If this intervention is fully active with probability of 1, the method is the same as an arc cut. The downside of adding intervention variables is that it will alter the original probability distribution over the unaugmented set of variables. That original distribution can be recovered by setting the priors over the new intervention variables to zero. Current BN software, lacking the support for explicit causal modelling, will not allow such variables to be instantiated. To solve this, one could maintain two BNs. One for reasoning about the effect of causal interventions and one for without [Korb and Nicholson, 2003]. In Figure 5 the modelling of causal interventions is demonstrated.
3.2.4 Decision networks

Decisions are made based upon the outcomes of the possible actions. Utility theory provides a way to assign utility (or value) to such outcomes. Actions can now be assessed so that the action with the maximized expected utility can be chosen. The combination of utility theory and probabilistic reasoning gives a framework for decision making. Decision networks (or Influence Diagrams) extends a BN with decision nodes and utility nodes to support decision making.

3.2.5 Knowledge engineering with Bayesian Networks

The construction of a BN can be generalized into a methodology for the development and deployment of BNs. The major modelling tasks for the development of a BN are [Korb and Nicholson, 2003]:

1. What are the variables? What are their values/states?
2. What is the graph structure?
3. What are the parameters (probabilities)?
And additional tasks for building Decision Networks:

1. What are the available actions/decisions, and what impact do they have?
2. What are the utility nodes and their dependencies?
3. What are the preferences?

Most of these tasks can be performed using expert elicitation. Machine learning, alone or in a combination with elicitation, can also be used for some tasks.

The process of constructing a BN can be compared to the software engineering process. Different life-cycle models exist and can be used to identify different phases in the process, see for example Figure 6. For the development of BNs a prototyping life-cycle seems very useful. Prototyping allows the organic growth of software, by stages from childhood to adulthood. Every stage results in a prototype which is a functional implementation of a BN. The prototype can easily be validated by an expert for the correct behaviour, because of the graphical aid of BNs.

---

**Variable and values** To keep the knowledge engineering task manageable, it is important to determine the importance of variables. Then, in the beginning of the knowledge engineering process, the number of variables/nodes can be limited to only the most important. Roles of variables/nodes that can be identified:

- Target or query nodes. The “output” nodes, whose values the end-users wants to evaluate.
- Evidence or observation nodes. The “input” nodes, resembling evidence that can be observed.
- Context variables, describing sensing conditions and background causal conditions.
- Controllable variables, that can be set by intervention in the domain environment.

Roles of variables/nodes can change, depending on how the BN is used. It is often useful to begin with identifying the query variables and then spread out to the related variables.

Most BN software tools require that variables take on discrete values. In most cases building a BN will be simpler if variables are discretized. Continuous variables can be discretized by converting them to multinomial variables where each value represents a different subrange of the original range of continuous values. Discrete values have to be exhaustive and exclusive, each variable must take on exactly one value at a time. Modelling variables with non-exhaustive or non-exclusive can result in a representation that does not represent reflect the real-life situation. 

---

Figure 6: Knowledge engineering with Bayesian Networks life-cycles: (a) Waterfall, (b) Spiral. 

[Korb and Nicholson, 2003]
Graphical structure  When building the graphical model a tradeoff has to be made between
the simplification of belief updating and between the accuracy of the model. Simplification means
to minimalize the nodes, arcs and the number of parameters but accuracy requires often more
nodes, arcs and parameters. An appropriate structure between these goals has to be found.
The different relationships between variables help in determining the appropriate structure for a
domain. The most important relationship is the causal relationship. Orienting arcs in a causal
direction is not required but it can maximize the representation of conditional independence.
This can lead to a more compact and simpler model. Dependence and independence relationships
also determine the model structure. If variables are dependent on each other the question is if
the variables should be directly connected or not, expressing a conditional dependency through
other variables. D-separation can be help in checking a structure with the problem domain.
An association relationship is a less explicit indicator for a model structure. It indicates that
knowing one variable provides information about another variable. Sometimes, there can also be
a relationship in the temporal ordering of variables. When building a causal network, temporal
information can restrict the orientation of some arcs. [Korb and Nicholson, 2003]

Probabilities  The probability assessment parameters are a set of conditional probability dis-
tributions of child values given values of parents. There is one distribution for each possible
instantiation of parent variables. The task of probability assessment is exponential in the num-
ber of parent variables, unless there is local structure reducing the parameters to be estimated.
There are three parameter sources:

- Learning parameters from domain data using data mining techniques.

- Eliciting process with Domain Experts. A difficulty here is that humans almost always
display various kinds of bias in estimating probabilities:
  - Overconfidence; the tendency to attribute higher than justifiable probabilities to
events that have a probability sufficiently greater than 0.5.
  - Anchoring; the tendency for subsequent estimates to be “weighed down” by an initial
    estimate.
  - Availability; assessing an event as more probable than justifiable, because it is easily
    remembered or more salient.

- Literature; There may be a publishes body of knowledge about the application domain.
  One common problem with published statistics is sparseness. For example, information
about the frequency of a symptom occurring when a disease is present is given, but not
the frequency of the symptom occurring when the disease is not present. [Korb and Nicholson, 2003]

Local structure  When the conditional probabilities to be learned not just depend upon the
parents variables’ values but also upon each other, there is local structure. Local structure are
methods for speeding up the learning of parameters for conditional probability distributions.
The speed up is achieved by reducing the parameters to consider. Examples of these methods
are Noisy-or, Classification graphs and Logit models. Another way to reduce the parameters
is by changing the graph structure. With divorcing multiple parents an intermediate node is
introduced that summarizes the effect of a subset of parents on a child. Thus the total amount
of parents on a child is reduced. [Korb and Nicholson, 2003]

Adaptation  Adaptation is adapting a BN’s structure or its parameters to (newly available)
data by using machine learning procedures. For the adaptation of multinomial parameters dif-
f erent algorithms exist, which i will not describe here. For the adaptation of the graph structure
no handy methods exist. A suggestion is the approach of accumulating cases and rerunning
structure learning algorithms in batch mode periodically. [Korb and Nicholson, 2003]
3.3 Causal prediction models in literature

In this section I will describe prediction models that use the Bayesian Network technique. The models all try to support in decision making for quality aspects in software engineering. The models follow the same idea for modelling the causality found in the software engineering process. However, different categories can be identified in which the models try to support in. The categories which I identified are: identifying uncertainty [Ziv and Richardson, 1997b], software quality [Gurp and Bosch, 2000; Fenton et al., 2007; Pai et al., 2005], software reliability [Gran and Heiminen, 2001; Marquez et al., 2007], engineering resources/effort [Bibi and Stamelos, 2004; Hearty et al., 2007; Hearty et al., 2009; Stamelos et al., 2003], or the development process with resulting software [Radlinski et al., 2007; Settas et al., 2006]. I will elaborate on two of these models to show how BNs can support in decision making.

3.3.1 Predicting software defects in varying development lifecycles

In [Fenton et al., 2007] Fenton et al. propose a causal model for defect prediction. Their model relates to a more general software project management aspect of software engineering compared to other models. The intention of this model is to help project managers to decide when to stop testing and release software.

The model is designed by using the causal understanding of project managers, rather than relying only on data from previous projects as is done with regression-based models (see section 3.4 Software reliability modelling). The probability tables of the BN are constructed using empirical data. For missing data or data that does not take into account all the causal influences, expert judgement is used.

Figure 7 shows the causal model for predicting the number of software defects. ‘Residual defects post’ has a relation with ‘defects’ in and ‘defects fixed’. This is based on the fact that the number of defects in and the number of defects fixed cause the number of residual defects. The number of defects in is based on the relation with the number of ‘potential defects given the adequacy of the specification and documentation’, which in turn is based on the number of thousands (Kilo) of Lines Of Code written and the probability of avoiding defects in development. The values for the input nodes, the dotted white ellipses, are determined through the help of three activity classes which correspond to activities that make up the software development lifecycle: Specification and Documentation, Design and Development, Testing and Rework Figure 8. For each activity, there is a different BN model with an output node that matches with an input node of the prediction model.

The division between the defect prediction model and the classes has been made so the BNs can be matched to different development lifecycles and the different phases in it. In a waterfall lifecycle each activity corresponds to a phase in the lifecycle. But for a incremental lifecycle all activities are done during a phase. This is mimicked in the prediction model by using the defect prediction output as an extra input in subsequent phases. See Figure 9 for the change in the prediction model. ‘Total defects’ takes the place of ‘Defects in’ and is a accumulation of the ‘New defects’ in (the former ‘Defects in’) and the ‘Residual defects pre’ (the defect prediction output of the previous phase).

The linking of the activity classes can be seen in Figure 10. Phases can be added by linking the Residual defects post to Residual defects pre in a new phase. Quality of spec and doc PRE and POST can be used in the same way. By linking phases, the prediction model can follow the phases in the development lifecycle.

3.3.2 Is my software “good enough” to release?

Donohue and Dugan propose in [Donohue et al., 2005] their Good Enough To Release(GETR) methodology to provide a probabilistic assessment of the acceptability of software quality. GETR assesses the overall quality of software preparatory to its release through the evaluation of process and product evidence. The methodology has three main elements: a model whose elements represent activities and artifacts identified as being effective indicators of software quality, processes for populating the model (the CPT’s), and methods for analyzing the contributions of
individual evidence. The first element, the model, is the GETR BN, see Figure 11. With the BN a probabilistic assessment of software’s overall “good enough” to release” quality is made. For the process of populating the model, the authors developed the Quantifying Judgement method for BBN(QJ-BBN) that transforms expert opinion about the contributions of certain activities into subjective conditional probabilities.

Analyzing the contributions of individual evidence of the nodes in the GETR BN is done with the authors version of a traditional BN sensitivity measure: the Contribution Importance Measure(CIM). With this Contribution Importance Analysis one can determine which activities or artifacts need extra effort to achieve a higher $Pr(\text{evidence} = \text{Acceptable})$.

### 3.4 Software reliability modelling

Support in decision making for quality aspects in software engineering can also be achieved through Software Reliability Growth Models(SRGM). Such models attempt to statistically correlate software reliability - through defect detection data - with known functions, such as an exponential function. If the correlation is good, the known function can be used to predict future behavior of the software system. Most SRGMs have a parameter that relates to the total number of defects contained in a set of code. If we know this parameter and the current number of defects discovered, we know how many defects remain in the code. Knowing the number of residual defects helps to decide whether or not the code is ready to release and how much more testing is required if the code is not ready to release. The optimal release time can be obtained
Figure 8: The Testing and Rework class models the quality of the testing and rework process. The better the testing process the more likely it is to find defects. Managers can decide to fix found defects in the testing phase, the success of such fixes will depend on the probability of fixing defects. The two probabilities of finding and fixing defects are used to update the corresponding nodes in Figure 7. [Fenton et al., 2007]

Based on a cost criterion when minimizing the total expected cost.

McDaid and Wilson [Mcdaid and Wilson, 2001] research the “deciding how long to test software” question by adopting a decision theoretic approach which has three components: a testing plan, a utility to describe the consequences of testing to a given time, and a probability model to describe the discovery of bugs over time. This probability model is a SRGM based on a non-homogeneous Poisson process. The optimal time to test is that which maximizes the expected utility. The utility function is build from the following assumptions:

1. There is a profit $A$ representing the income that the producers of the software would obtain if the program were to be released immediately and to contain no bugs.

2. The cost of repairing each bug found during the testing phase is a constant $C$.

3. The cost of discovering a bug after testing (i.e. when the software is in service) is a constant $D$; typically, $D$ is much larger than $C$.

4. The cost of testing the program to time $T$ and the lost opportunity, in terms of lost sales and loss of market initiative, are expressed by a function $F(T)$. A simple form for $F(T)$ would be linear,

$$ F(T) = (S + M + R)T $$

where $S$ is the staff cost, $M$ the machine cost and $R$ the lost revenue per unit of time.
Combining these assumptions, there is a utility function for testing software for a time $T$, in which $N(T)$ bugs are discovered and corrected, followed by release where $\overline{N}(T)$ bugs arise:

$$U\{T, N(T), \overline{N}(T)\} = A - CN(T) - D\overline{N}(T) - F(T)$$

With this utility function McDaid and Wilson can calculate the time to test software that maximizes the profit. The calculation can be made for a single-stage testing process to a multi-stage testing process. In a single-stage there is one testing phase and after its completion the software is released. In a multi-stage testing process there can be more testing phases and after each testing phase the decision is made to release the software or to start another testing phase. The advantage of a multi-stage is that after each phase there is an opportunity to adjust the input of
Figure 11: The ‘Good Enough’ To Release BN. The nodes are all discrete with the states being \{Acceptable, Unacceptable, Unknown/Uncertain\}. The ‘Good Enough To Release’ node is determined by Process and Product evidence. Product evidence includes artifacts of validation and verification activities and the documentation of the satisfaction of specified release criteria. Process evidence includes judgments as to the quality of various aspects of the development process, such as the skill sets of development personnel, change management policies, and project management. [Donohue et al., 2005]
the SRGM. The disadvantage is that allowing unlimited testing phases means that the decision problem cannot be solved.

See [Huang and Lyu, 2005] for a research about incorporating a logistic testing-effort function into software reliability modelling. They also present a software cost model to reflect the effectiveness of introducing new testing technologies or tools for increasing the efficiency of software testing.

4 Research approach

This research will develop a prediction model that supports cost-benefit analysis of testing inside the software factory Endeavour. The research can be divided into two parts. The first part will be research into the problem domain: the needed information for cost-benefit analysis of testing. The second part will be research on how to create a prediction model for the problem domain. These parts do not necessarily have to follow each other in the research. They are researched at the same time as the practical way to engineer a Bayesian network is to use a spiral model discipline.

Part one The first part of the research is looking into what the requirements are for cost-benefit analysis of software testing. This will start by researching the software factory developed by the IT company where this research will be conducted. The software factory is a collection of software engineering project experience combined into guidelines and processes stating best-practices. The company uses these best-practices to control the software engineering process. This way they know, to some amount, how much resources are necessary for a project and how long a project will take. Ensuring a certain product quality as the outcome of a project is an ultimate goal for a software factory. The software factory uses metrics to measure different aspects of the software engineering process. These metrics are stored for every project executed in the factory. To this day there is for multiple years of experience compiled into the factory and the factory has been set up to accommodate software projects for the same type of software product, namely administrative software based on a Service Oriented Architecture in a project based on the Rational Unified Process.

The research on the software factory will result in an overview of the used testing processes. This overview will consist of the factors that influence the testing process and the factors that determine the successfulness of a testing process. The research will also look into which factors uncertainty plays a role and describe this. At the same time the research will compare the findings with a literature study about the testing processes used in the software factory.

The research will also consist of a literature study to prediction models that incorporate variables from the testing process domain. This study will describe known correlations in software engineering for the prediction of testing related information.

The goal of part one is a description of the variables that influence testing, the variables that are influenced by testing and the relations between them.

Part two The second part is the creation of the prediction model and measuring the probability estimating capabilities of it. With the outcome of part one, there is enough information to build the prediction model.

The created model will be a Bayesian network, which is a causal network that makes reasoning about the variables and uncertainty in the network possible. Before creating the models, research will be done on how to create BNs, how to train them and on what a plausible good design can be for the prediction model.

The model will be trained with empirical project data available in the software factory through metrics and expert judgements. To measure the performance a test set will be created. The test set will consist of empirical project data from X number of projects that where not used for the training of the model. The model performance will be measured a as a probability estimator on the projects in the test set.
The conclusion of part two describes a prediction model for cost-benefit analysis, the choices made in the creation of that model, and the performance of the model as probability estimator.

Figure 12: An Bayesian Network example of a model for cost-benefit analysis on testing. It shows a relation between the effort put into testing and the number of found pre-release and post-release defects. The project size influences the number of defects found.

5 Research objective

The general objective of this study is to contribute to the development of theory regarding metrics, in the form of Bayesian Network prediction models. This study will take into account the idea that a causal explanatory prediction model that incorporates uncertainty provides better risk assessment.

A secondary objective of this study is to contribute to the knowledge of Info Support, the IT company in which this study takes place. The contribution is knowledge about testing risk assessment in the form of a causal explanatory prediction model for testing efficiency.

6 Planning

See Figure 13. The goal for the research questions is also to write down the findings in the thesis. So the thesis will be incrementally written. The draft MSc. Thesis goal is to finalize a draft thesis that can be used for reviewing. The black X’s indicate the main goal for that week. A gray X indicates possible effort on that goal.

References


<table>
<thead>
<tr>
<th>Week number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task B</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task C</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task D</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task E</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task J</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Task R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Figure 13: Research planning.**

**Final MSCT analysis.**

**Draft MSCT analysis.**

**Part 2: Model creation.**

**Part 1: Cost-benefit analysis.**

**Research question.**

**Goal.**