Using data flow analysis for the reliability assessment of safety-critical software systems

BÖRCSÖK J., SCHAEFER S.
Department of Computer Architecture and System Programming
University Kassel,
Wilhelmshöher Allee 71, D-34121 Kassel
GERMANY
j.boercsoek@uni-kassel.de schaefer@uni-kassel.de www.rs.eecs.uni-kassel.de

Abstract: Reliability analysis for safety-critical software systems often needs additional expert knowledge, because of the small data-sets available. A Bayesian approach is used to develop a reliability model based on expert knowledge and small data-sets. The expert knowledge is obtained with data flow analysis. Certain variables in the program code are examined to calculate their probability of causing a failure. These additional information are incorporated into a suitable distribution function to be able calculate reliability characteristics with greater precision.

Key-Words: software reliability, bayesian reliability, static analysis, value analysis, abstract interpretation

1 Introduction

1.1 Motivation
Safety-critical software has to be developed according to standards and has to be certified. Therefore it is necessary to have exact calculations of the required reliability parameters. The usual software reliability models use failure data-sets to estimate the parameters of the underlying model. This method is often not suitable for safety-critical software, because of the small data-sets that are available. Safety-critical software has a high degree of maturity, when data collection begins, so that only a few failures if any can be recorded.

1.2 Requiring additional information
Therefore it is necessary to develop a reliability model that accounts for this lack of data. A reliability model for safety-critical systems requires therefore additional information. The additional information can be derived from different areas of the software development process. Experience from past projects can be integrated into the model, data from software projects that behave similar or are developed under similar conditions is existent, expert knowledge is available, that was generated from code reviews and inspections or the software can be subjected to static analysis. Because of the high complexity of modern safety-critical software systems e.g. the source code of modern infusion pumps can be comprised of up to 170,000 lines of code [6], complete code reviews are not feasible. On the other hand it is not possible to use data from former or similar projects, because even small changes e.g. in the operational profile makes it impossible to extrapolate to the project that is examined.

1.3 Related Work
A comprehensive study of software reliability models and the underlying mathematical structures is given in [7] and [8], although the special conditions of safety-critical software is not taken into considerations in these works. [9] examines different reliability models on the basis of scarce failure data from safety-critical space shuttle software, but does not incorporate any extra information into the calculations.

Static analysis is in [1] applied to safety-critical software, but is mostly used to examine certain characteristics of the software rather than use it to calculate its reliability.

2 Static analysis
Static analysis has none of the aforementioned restrictions that are making it hard to use knowledge from past experience. It can be automatically applied to a large code basis and the results can processed automatically. Static analysis analyses the source code of the system, without executing it. For
our purpose its aim is to verify specific data dependent characteristics that can lead to faults in a running system and subsequently to failures. Typical data dependent faults are: Division by zero, Pointer analysis, Buffer overrungs or Value analysis. Because every non-trivial problem in a complex system is undecidable it is impossible to proof the aforementioned characteristics. This often results in a tradeoff between false positives and false negatives that has to be balanced.

In spite of this static analysis yields useful results. It gives probabilities for occurrence of certain kinds of faults that can be incorporated into the reliability model. With the use of abstract interpretation it is possible too derive sound results.

Value Analysis

With the use of abstract interpretation [2] it is possible to perform value analysis. Abstract interpretation allows approximating the concrete semantics of a system. The concrete semantics of a program describes all possible executions of this program. It is not possible to infer directly the required information from the program because all non-trivial characteristics are undecidable for the concrete semantics. The abstract interpretation constructs conservative approximations with a superset, the so-called abstract semantics from the concrete semantics, e.g. the abstract information of a certain integer value could be: odd, even, negative or the restriction to an interval. The advantage of abstract interpretation is due to its soundness, i.e. a property that is proven in the abstract semantic holds in the concrete semantics, thus avoiding false positives. In addition abstract interpretation is precise enough to avoid too many false negatives.

Abstract interpretation consists of two functions α and γ. The function α is called the abstraction function. It maps the concrete state of a date to an abstract value. The function γ is the concretization function. It maps the concrete state of a date to an abstract value. The functions α and γ form a galois connection:

\[ S \subseteq \gamma(a) \iff \alpha(S) \subseteq a \tag{1} \]

where \( S \) is a concrete state and \( a \) is a abstract property, e.g. negative/non-negative. The galois connection allows the abstract interpretation to be exact, because if α and γ are monotone (1) yields:

\[ S \subseteq \gamma \circ \alpha(S) \quad \text{and} \quad \alpha \circ \gamma(a) \subseteq a \tag{2} \]

with the restriction \( a = \gamma(a) \) for the right term of (2) this gives proof that an abstract interpretation is exact.

The value analysis tries to compute all possible values for a program variable for every element of the concrete semantics with abstract interpretation. There are two basic approaches:

1. Constant propagation: The exact value of the variable is known or there is no information at all for this variable.
2. Interval analysis: The abstract property \( a \) is defined as an interval. The value of the variable is within the range of this interval or it is not.

[4]) describes a general framework for interval analysis. In [10] an implementation for the general framework is given. This implementation is used as an example in this paper to demonstrate the feasibility of this approach. An own implementation with possible improvements is intended for future work.

The implementation in [10] uses forward and backward propagation to have more precise results, which means that the source code is analyzed twice. The first iteration explores the code from start to end and the second iteration from end to start. An example transfer function is given to illustrate the value analysis. An interval for an variable \( x \) is given as \(<x_\text{min}, x_\text{max}>\). Possible operations on sets are defined, like unions and intersections, e.g.

\[ <x_1, x_2> \cap <y_1, y_2> = [\min(x_1, y_1), \max(x_2, y_2)] \]

An addition in the concrete semantics is then computable in the abstract domain:

\[ x = y + z \in S \rightarrow \]

\[ y \uparrow = y \uparrow \cap <x_1 - z_\text{min}, x_1 - z_\text{max}> \in \alpha(S) \]

\[ z \uparrow = z \uparrow \cap <x_1 - y_\text{min}, x_1 - y_\text{max}> \in \alpha(S) \tag{4} \]

The direction of the arrow describes if forward or backward propagation is computed. The transfer functions for other operations in the concrete domain are obtained in a similar fashion.

For every variable for which this is requested a value analysis can be done. The interval that is examined is here the range that is representable on the given system. On a 32 bit system the resulting interval for an unsigned integer is \(<0, 2^{32}>\). If the variable does not exceed or underruns this interval in the abstract domain no underflow or underflow is possible. Thus this variable can not cause a fault in any program state. After the static analysis information is available which variables are safe, i.e. can never cause faults and which variables are unsafe, i.e. they can cause faults but do not necessarily cause problems.
Usually it is too expensive to execute value analysis for every program variable. The software or reliability engineer has to flag the variables that are safety critical and have to be examined.

The resulting information is then incorporated into the reliability model as prior information, which is independent from the registered failure-data.

3 Bayesian reliability

3.1 Principles

The common approach to add extra information into a reliability model is through bayesian reliability. The difference to the classic frequentist approach is that the Bayesian approach yields a confidence interval for the estimated parameters of the reliability model. This gives additional certainty in the calculated values for failure rate or failure intensity.

The Bayesian approach is based upon four density functions with the following relation:

$$g(\lambda | x) = \frac{f(x | \lambda)g(\lambda)}{\int_0^{\infty} f(x | \lambda)g(\lambda)d\lambda}$$ (5)

where $g(\lambda | x)$ is the posteriori distribution. This is the fundamental distribution for the software reliability model. The parameters are $x$ as the collected data, in this case the recorded failure times and $\lambda$ as the estimated parameters of the model. The posteriori distribution gives the most likely parameters given the recorded failure times. The prior distribution is given by $g(\lambda)$. This the initial estimate of the parameters before any failure times are collected. The form of the prior distribution is chosen because of the underlying constraints or because of mathematical convenience. The prior distribution contains the information from the static analysis. The function $f(x | \lambda)$ is called the likelihood function. It describes the probability of the occurrence of the collected data given the parameter of the model. The denominator of equation (5) is called the marginal distribution and it represents a normalizing factor for the posteriori distribution. A complete theoretical background of Bayesian reliability is given in [3].

3.2 Discrete Case

In the discrete case the software that is to be analyzed has only discrete runs. It is a piece of code that will be executed on demand and performs a specific function, e.g. safety measures. It will then be suspended until the next demand of the software arises. The interesting parameter is the probability that this software module will fail on demand. The parameters $\lambda$ and $x$ in equation (5) can then be interpreted as parameters of bernoulli trials $p$. The parameter $p$ denotes the probability that an event occurs. Here an event is the failure of the software. The number of trials that have successful outcomes is represented by $k$.

The software is tested with a certain amount of runs $n$. The trials where an event, i.e. a failure, has occurred are recorded and can be used to directly estimate $p$ or to use these parameters as input for the likelihood function. For the latter case the likelihood function is regarded as a binomial distribution

$$f(p | k) = \Pr(K = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$ (6)

Figure 1 shows two binomial distributions with different probabilities.

The direct consideration of the number of runs of the software is often not possible, because for safety-critical software there is not enough data available. Due to the fact that for safety-critical software only a limited number of failure times or even no failure times at all can be measured it is not possible to infer reliable model parameters from a direct maximum likelihood procedure. Furthermore a higher precision of the parameters is expected with the use of prior information.

![Fig. 1: Two examples of a binomial distribution](image)

3.3 Continuous case

The second approach considers the run of the software as continuous, i.e. there are no completed runs as in the previous section. That allows for a time dependent analysis so that time dependent
reliability characteristics like MTTF, failure rate or failure intensity can be calculated.

A distribution that is often used because of its flexibility and its mathematical tractability is the gamma distribution $Ga(a,b)$ as a prior distribution:

$$g(\lambda) = g(\lambda; a,b) = \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda} \quad (7)$$

The parameters of equation (7) are $a$ the so-called shape parameter and $b$ the rate parameter, and the function $\Gamma(a)$ denotes the gamma function. The flexibility stems from the fact that, with the gamma distribution it is possible to model increasing, decreasing and constant failure rates. For $a=1$ the gamma distribution becomes the exponential distribution which is used to model constant failure rates. If the shape parameter is $a<1$ the resulting failure rates are decreasing and for $a>1$ the failures are increasing.

The parameters $a$ and $b$ have to be estimated based on the value analysis. In order to do that an assumption, regarding the software life cycle, to determine parameter $a$ has to be made. If every error that caused a failure is found and fixed without introducing new errors or if the error-introducing rate is smaller than the fixing rate the failure rate of the software over its life time is decreasing and therefore the shape parameter has to be smaller than 1. Accordingly if the software underlies software entropy the code becomes less reliable and this results in increasing failure rates and $a>1$. If the failures are only recorded, but no error-fixing are performed the code basis is unchanged and $a=1$ with a constant failure rate. [5]

The parameter $b$ represents the rate of the distribution. As an initial estimate the ratio of unsafe to safe variables can be used. The practicality of this ratio can be improved by using the number of executions of unsafe and safe variables as underlying ratio. Since not every execution of an unsafe variable leads to an erroneous program state, it is necessary to use a proportionality factor that reflects the probability that an execution of an unsafe variable results in an error. This proportionality factor can be found through empirical data or with expert knowledge.

4 Application

4.1 Application of the discrete case

The value analysis gives information about a fraction of variables in the code that can cause errors and subsequently failures, i.e. every time a variable that was flagged unsafe is executed there is a small probability of a failure. This behavior can be used to model the prior distribution of equation (5). For the form of the prior distribution usually a distribution is chosen that belongs to the same family as the posterior distribution. The prior distribution is then called a conjugate prior in respect to the likelihood. The beta distribution can be used as a conjugate prior for the binomial distributed likelihood function in equation (6):

$$g(p) = Be(a,b) = \frac{\Gamma(a + b)}{\Gamma(a) \Gamma(b)} p^{a-1} (1-p)^{b-1} \quad (8)$$

The parameters $a$ and $b$ of this distribution are chosen as the ratio of unsafe $v_u$ to overall variables $v_a$ and vice versa. The rationale behind this is as follows. The ratio $a=v_u/v_a$ describes the initial trust into the system on the basis of the static analysis. Even with no actual run of the software the probability of failure is small when there are only a small number of unsafe variables, which could cause these failures, in comparison to the overall number of variables. The second parameter is used in a similar fashion. The ratio $b=v_a/v_u$ describes the statistical spread of the distribution. This ratio is supposed to be large, which in turn makes the spread smaller. The fact that both parameters are dependent of each other can be neglected, because this specific prior distribution is chosen due to its mathematically convenience and only one parameter is needed.

Equations (6) and (8) are used to derive the posterior distribution. Then equation (5) becomes:
g(p | a, b, k) = \frac{\binom{n}{k} p^k (1 - p)^{n-k} Be(a,b)}{\int_0^n \binom{n}{k} p^k (1 - p)^{n-k} Be(a,b) dp} \quad (9)

After integration the denominator in (9) gives the beta-binomial distribution and with transformation and simplifying this results in:

\[ g(p | x') = \frac{\Gamma(n+a+b)}{\Gamma(k+a)\Gamma(n+b-k)} p^{k_1-1} (1-p)^{n+b-k_1} \]
\[ g(p | x') = Be(k+a), (n+b-k) \quad (10) \]

with \( x' \) as the set of parameters \( (a,b,k) \).

4.1.1 Example

On-demand software is tested 100 times. Two failures are recorded. The resulting MLE for the corresponding binomial distribution yields \( p = k/n = 0.02 \). The mean and the standard deviation of the classical approach to estimate \( p \) are \( \mu = 2 \) and \( \sigma^2 = 1.4 \). The software consists of 1000 variables that are analyzed, for which 10 of these variables are flagged as potentially unsafe. The resulting initial prior Beta distribution \( g(p) = Be(0.01,100) \). For the posteriori distribution follows \( g(p|x') = Be(2.01,198) \). The mean and the standard deviation for the bayesian approach with static analysis are \( \mu = 0.01 \) and \( \sigma^2 = 0.07 \).

4.1.2 Special Case

If the on-demand software is tested 100 times and no failure is recorded it is not possible to calculate a meaningful probability for the binomial distribution, because \( p = 0 \) and therefore \( \mu = 0 \) and \( \sigma^2 = 0 \), which implies error-free software which is not realistic assumption. If the bayesian method is used with the same prior distribution as in the above section the resulting distribution has meaningful parameters \( \mu = 0.00005 \) and \( \sigma^2 = 0.0053 \), which are usable to determine reliability characteristics.

4.2 Application of the continuous case

In the continuous case the prior distribution is modeled by the gamma distribution. The gamma distribution can act as the conjugate prior for different likelihood functions. A widely used distribution in reliability engineering is the exponential function \( f Ex(\lambda) = \lambda e^{-\lambda t} \). The use of the gamma distribution \( Ga(a,b) \) as prior and of the exponential \( Ex(\lambda) \) as likelihood yields the following gamma distributed posteriori distribution:

\[ g(\lambda | a, b, n, T) = \frac{(b+T)^{a+n}}{\Gamma(a+n)} \lambda^{a+n-1} e^{-(b+T)\lambda} \]
\[ g(\lambda | a, b, n, T) = Ga(a+n, b+T) \quad (11) \]

where \( n \) and \( T \) consists of the information that comes from testing the software, \( n \) is the number of failures that are recorded and \( T \) is total test time or the sum of failure times \( t_i \).

The parameter \( a \) of the prior distribution is established through the assumption on the software life-cycle. An increasing failure rate is assumed with \( a = 1.5 \). Parameter \( b \) takes the value of the ratio of the safe and unsafe variables that are analyzed in the value analysis step.

4.2.1 Example

The example of the section 4.1.1 is used again, but the software is tested for 100 hours, instead of 100 trials. Two failures are recorded at 30 hours and 60 hours. If the exponential model is used directly with \( n = 2 \) and \( T = 30+60 \), the maximum likelihood estimate for \( \lambda \) is \( n/T = 0.022 \). This can be interpreted as the failure rate of the system. The mean, which is the mean time to failure (MTTF) and the standard deviation are \( \mu = MTTF = 45 \) and \( \sigma^2 = 45 \). For the Bayesian approach the number of analyzed variables that are safe is 1000 and 10 unsafe variables are found. The resulting posterior distribution is \( Ga(3.5,190) \), with \( MTTF = 665 \) and \( \sigma^2 = 0.0098 \).

4.2.2 Special Case

Analog to the discrete case there is no possibility to calculate meaningful values for the MTTF for the exponential distribution if there are no failures recorded. The prior distribution can be used to estimate parameter, which can then be refined with the likelihood function even when no further failures occur.

5 Conclusion

The example calculations for the discrete and the continuous case show promising results. In both cases the estimated reliability is larger within the bayesian approach. The trust in these results is furthermore increased as the standard deviations are smaller, when using the bayesian method with static analysis.

The classic methods fail to produce useful reliability measurements when no failure data is recorded, which is often the case, when testing software for safety critical systems. The value analysis allows making reasonable estimates for
prior distributions so that useful reliability characteristics can be calculated.

6 Future Work
The theoretical considerations have to be verified with real systems. It is necessary to investigate failure data that stems from safety-critical software and to compare these with the synthetic results. For this purpose an implementation of the value analysis is necessary that can analyze the software in an efficient and precise way. With the achieved results the parameter of the prior distributions can be refined to gain more precise reliability characteristics.

References