Utilizing Verification and Validation Certificates to Estimate Software Defect Density

Mark Sherriff
North Carolina State University
Raleigh, NC, USA 27695
+1-919-513-5082
mark.sherriff@ncsu.edu

ABSTRACT

In industry, information on defect density of a product tends to become available too late in the software development process to affordably guide corrective actions. An important step towards remediation of the problem associated with this late information lies in the ability to provide an early estimation of defect density. Our research objective is to build a parametric model which utilizes a persistent record of the validation and verification (V&V) practices used with a program to estimate the defect density of that program. The persistent record of the V&V practices are recorded as certificates which are automatically recorded and maintained with the code.

PhD Advisor: Dr. Laurie Williams, williams@csc.ncsu.edu

Categories and Subject Descriptors

D.2.8 **[Software Engineering]:** Metrics - *Performance measures*, *Process metrics*, *Product metrics.*

General Terms

Measurement, Design, Reliability

Keywords

Software Reliability Engineering, Reliability Estimation, Validation and Verification Management

1. INTRODUCTION

In industry, post-release defect density of a software system cannot be measured until the system has been put into production and has been used extensively by end users. The defect density of a software system is calculated by measuring the number of failures divided by the size of the system, using a size measure such as lines of code. Actual post-release defect density information becomes available too late in the software lifecycle to affordably guide corrective actions to software quality. Correcting software defects is significantly more expensive when the defects are discovered by an end user compared with earlier in the development process [4].

Because of this increasing cost of correcting defects, software developers can benefit from early estimates of the defect density of their product. If developers can be presented with this defect density information during the development process in the environment where they are creating the system, more affordable corrective action can be taken to rectify defect density concerns as they appear.

A development team will use several different methods to ensure that a system is of high-assurance [14]. However, the verification and validation (V&V) practices used to make a system reliable might not always be documented or this documentation may not be maintained. This lack of documentation can hinder other developers from knowing what V&V practices have been performed on a given section of code. Further, if code is being reused from an earlier project or code base, developers might spend extra time re-verifying a section of code that has already been verified thoroughly.

Research has shown that parametric models [6] using software metrics, such as the Software Testing and Reliability Early Warning (STREW) [10] suite, can be an effective means to predict product quality. *Our research objective is to build a parametric model which utilizes a persistent record of the V&V practices used with a program to estimate the defect density of that program.* To accomplish this objective, we are developing a method called Defect Estimation with V&V Certificates on Programming (DevCOP). This method includes a mechanism for creating a persistent record of V&V practices as *certificates* stored with the code base, a parametric model to provide an estimate of defect density, and tool support to make this method accessible for developers. A DevCOP certificate is used to track and maintain the relationship between code and the evidence of the V&V technique used. We will build the parametric model using a nine-step systematic methodology for building software engineering parametric models [2], which has been used to build other successful parametric models [3, 10, 12].

2. BACKGROUND

In this section, we will discuss the relevant background work and methodologies used during our research. It includes descriptions of research regarding other metric-based defect density estimation, V&V techniques, and parametric modeling in software engineering.

2.1 Parametric Modeling

Parametric models relate dependent variables to one or more independent variables based on statistical relationships to provide an estimate of the dependent variable with regards to previous data [6]. The general purpose of creating a parametric model in software engineering is to help provide an estimated answer to a software development question early in the process so that development efforts can be directed accordingly. The software development question could relate to what the costs are in creating a piece of software, how reliable a system will be, or any number of other topics.

Parametric modeling has been recognized by industry and government as an effective means to provide an estimate for project cost and software reliability. The Department of Defense, along with the International Society of Parametric Analysts, acknowledges the benefit of using parametric analysis, and encourages their use when creating proposals for the government [6]. The Department of Defense claims that parametric modeling has reduced government costs and also improved proposal evaluation time [6]. Boehm developed the Constructive Cost Model (COCOMO) [3] to estimate project cost, resources, and schedule. Further, the Constructive Quality Model (COQUALMO) added defect introduction and defect removal parameters to the COCOMO to help predict potential defect density in a system. Nagappan [10] created a parametric model with his Software Testing Reliability Early Warning (STREW) metric suite to create an estimate of failure density based on a set of software testing metrics. In our research, we will also build a parametric model to estimate defect density based upon V&V certificates recorded with the code.

2.2 Verification and Validation Techniques

During the creation of software, a development team can employ various V&V practices to improve the quality of the software [1]. For example, different forms of software testing could be used to validate and verify various parts of a system under development. Sections of code can be written such that they can be automatically proven correct via an external theorem prover [14]. A section of a program that can be logically or mathematically proven correct could be considered more reliable than a section that has “just” been tested for correctness.

Other V&V practices and techniques require more manual intervention and facilitation. For instance, formal code inspections [5] are often used by development teams to evaluate, review, and confirm that a section of code has been written properly and works correctly. Pair programmers [15] benefit from having another person review the code as it is written. Some code might also be based on technical documentation or algorithms that have been previously published, such as white papers, algorithms, or departmental technical reports. These manual practices, while they might not be as reliable as more automatic practices due to the higher likelihood of human error, still provide valuable input on the reliability of a system.

The extent of V&V practices used in a development effort can provide information about the estimated defect density of the software prior to product release. The Programatica team at the Oregon Graduate Institute at the Oregon Health and Science University (OGI/OHSU) is working on a method for high-assurance software development [14]. Programmers can create different types of *certificates* on sections of code based on the V&V technique used by the development on that section of the code. Certificates are used to track and maintain the relationship between code and the evidence of the V&V technique used. Currently, the three types of V&V techniques that Programatica can create certificates for include expert opinion, unit testing, and formal proof. These certificates are used as evidence that V&V techniques were used to make a high-assurance system [14]. We propose an extension of OGI/OHSU’s certificates for defect density estimation whereby the estimate is based upon the effectiveness of the V&V practice (or lack thereof) used in code modules.

2.3 Metrics to Predict Defect Density

Operational profiles have been shown to be effective tools to guide testing and help ensure that a system is reliable [8]. An operational profile is “the set of operations [available in a system] and their probabilities of occurrence” as used by a customer in the normal use of the system [9]. However, operational profiles are perceived to add overhead to the software development process as the development team must define and maintain the set of operations and their probabilities of occurrence. Rivers and Vouk recognized that operational profile testing is not always performed when modern constraints on market and cost-driven constraints are introduced [11]. They performed research on evaluating non-operational testing and found that there is a positive correlation between field quality and testing efficiency. Testing efficiency describes the potential for a given test case to find faults at a given point during testing.

Nagappan [10] performed research on estimating failure density without operational profiles by calibrating a parametric model which uses in-process, static unit test metrics. This estimation provides early feedback to developers so that they can increase the testing effort, if necessary, to provide added confidence in the software. The STREW metric suite consists of static measures of the automated unit test suite and of some structural aspects of the implementation code. A two-phase structured experiment was carried out on 22 projects from junior/senior-level software engineering students from the fall of 2003 [10], which helped to refine the STREW-J metric suite. The refined suite was then used 27 open source Java projects found on SourceForge[1], an open-source development website, and five projects from a company in the United States [10]. The research from these case studies indicates that the STREW-J metrics can provide a means for estimating software reliability when testing reveals no failures.

Another version of the STREW metric suite was developed specifically for the Haskell programming language, STREW-H. STREW-H was similarly built and verified using case studies from open-source and industry. An open-source case study [13] provided guidance to refine the metric suite for its use on an industry project with Galois Connections, Inc [12]. These findings also showed that in-process metrics can be used as an early indicator of software defect density for Haskell programs.

3. PROPOSED SOLUTION

We propose a non-operational parametric model to estimate defect density based upon records of which V&V practices were performed on sections of code. We also wish to integrate our estimation directly into the development cycle so that corrective action to reduce defect density can take place early in the development process. We call this method the Defect Estimation from V&V Certification on Programming (DevCOP) method. A V&V certificate contains information on the V&V technique that was used to establish the certificate. Different V&V techniques will provide a different level of assurance as to how reliable a section of code is. For example, a desk check of code would be, in general, less effective than a formal proof of the same code.

We envision the defect density parametric model to take the form of Equation 1. For each certificate type, we would sum the product of a size measure (perhaps lines of code or number of functions/methods) and a coefficient produced via regression analysis of historical data. The calibration step of the regression analysis would yield the constant factor (a) and a coefficient weighting (c_j) for each certificate type, indicating the importance of a given V&V technique to an organization’s development process.

$$DefectDensity = a + \sum_{j=1}^{certificate\_type} (c_j * Size_j) \quad \textbf{(1)}$$

To build and verify our parametric model of our DevCOP method, we are utilizing the nine-step modeling methodology developed

[1] http://www.sf.net/

by the Center for Software Engineering at the University of Southern California [2]:

1. Determine model needs;
2. Analyze existing literature;
3. Perform behavioral analysis;
4. Define relative significance;
5. Gather expert opinion;
6. Formulate a priori model;
7. Gather and analyze project data;
8. Calibrate a posteriori model; and
9. Gather more data; refine model.

The goal of the model is to provide an estimate of defect density based on V&V certificates and the coefficient weights. We anticipate that a model would need to be developed for each programming language we would study. Our current work involves the Java (object-oriented) and Haskell (functional) languages.

The second step is to analyze existing literature to determine categories of V&V techniques and empirical findings on the defect removal efficacy of each V&V practice. Balci categorized V&V techniques with some regard to their general effectiveness as to finding defects in a system [1]. For the purposes of our scale, we began with Balci's categorization of the V&V techniques:

- Manual – includes all manual checking, such as pair programming [15] and code inspections [5];
- Static – includes automatic checking of code before run-time, such as syntax and static analysis;
- Dynamic – includes all automatic checking that takes place during execution, such as all forms of black-box testing;
- Symbolic – includes all model-based checking, such as path analysis;
- Constraint – includes all defined assertions and constraints programmed into the code base [14];
- Formal – includes all strictly mathematical forms of checking, such as lambda calculus and formal proofs [14].

Assigning proper relative significance to certificates to place them on a single scale of relative effectiveness is a significant challenge in our research. Each of these V&V categories provides different evidence as to how reliable a system is [1]. For example, static V&V techniques can provide information as to whether the structure of the code is correct, while manual V&V techniques can provide information about both the structure of the code and if the code is providing the functionality requested by the customer(s). We will perform a causal analysis with our industry partners on our initial data to help build our V&V rubric. The causal analysis will provide us with more information about the efficacy of certain V&V techniques under particular circumstances.

We must also determine the proper granularity for the model. Code certificates could potentially be associated with modules, classes, functions, or individual lines of code. Each level of granularity offers potentially different information about the defect density of the system, and also different challenges in gathering data. Currently, we are analyzing certificates at the function level and are involved in on-going analysis with our industry partners on this decision.

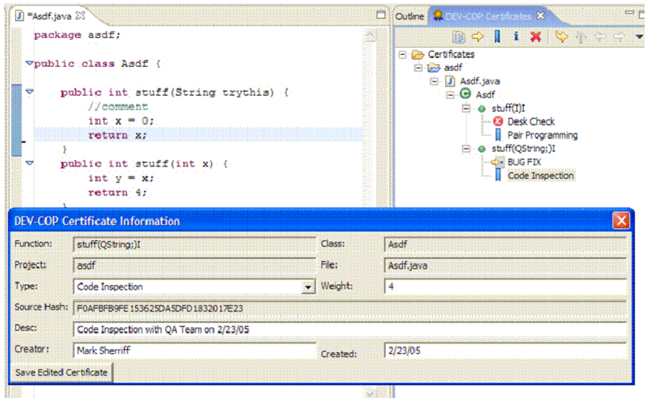
We are working with industry partners to gather expert opinion and our initial data sets. Developers on a small Java team using Eclipse are recording their V&V efforts using the DevCOP plug-in (described in Section 6) as the project progresses. During defect removal and bug fixes, the team will also record these efforts as a different type of certificate. Proceeding through steps 6-9 of the parametric modeling methodology will require a significant number of projects for each language we work with.

4. LIMITATIONS

In the creation of certificates, we are not assigning more importance to certain functions or sections of code over others, as is done with operational profile means of estimation. Nor are we using the severity of defects detected to affect the importance of some certificates over another. While this level of granularity could be beneficial, one of our initial goal's is to make this method easy to use during development, and at this time, we think that adding this level of information could be a hindrance. Another granularity limitation is the granularity of certificates. Based on the Programatica Team's work and expert opinion, it was decided that methods would be the proper level of granularity for certificates. As previously discussed, we recognize that being able to record certificates and a line of code level could be beneficial, but at this time method-level recording seemed to be the best course of action for the initial validation of the methodology.

5. TOOL SUPPORT

We will automate the DevCOP method with little additional overhead for developers. Ease of use, along with the added benefit of being able to calculate V&V and defect information with a defect density estimate, should make the DevCOP method practical for practicing engineers. We have created the first version of an DevCOP Eclipse[2] plug-in to handle the creation and management of V&V certificates during the development process[3][13]. The plug-in allows developers to create certificates during the development process within the integrated development environment (IDE) so that this information can be utilized throughout the code's lifetime. Figure 1 shows a screenshot of the Eclipse plug-in for recording V&V certificates.



```
package asdf;

public class Asdf {

    public int stuff(String trythis) {
        //comment
        int x = 0;
        return x;
    }
    public int stuff(int x) {
        int y = x;
        return 4;
```

Function:	stuff(QString;)I	Class:	Asdf
Project:	asdf	File:	Asdf.java
Type:	Code Inspection	Weight:	4
Source Hash:	F0AFBFB9FE153625DA5DFD1832017E23		
Desc:	Code Inspection with QA Team on 2/23/05		
Creator:	Mark Sherriff	Created:	2/23/05

Figure 1. Screenshot of the DevCOP Eclipse plug-in for recording V&V certificates.

In the current version of the plug in, programmers can select one or more functions for certification through the Eclipse Package

[2] For more information, go to http://www.eclipse.org/.

[3] The plug-in is available at http://arches.csc.ncsu.edu/sherriff/devcop/.

Explorer. They assign the type of certificate (i.e. Code Inspection, Pair Programming, Bug Fix) and the weight coefficient associated with it. The certificate information is then stored in an XML document that is saved in the project's workspace. The Eclipse plug-in reads and writes to this XML document as certificates are created and edited.

We have made the certificate creation process as easy and transparent as possible, and will continue to improve it in later iterations as we receive more developer feedback. For example, if two programmers were about to start pair programming on a piece of code, it would be beneficial if they could press a single button and the Eclipse plug-in would then mark all code until the button is pressed again as having a pair programming certificate. Enhancements such as this are currently under development.

6. CONCLUSIONS AND FUTURE WORK

We have created and are validating a method for managing and leveraging the effort put into V&V by a development team to provide an estimate of software defect density in-process. Due to the high costs of fixing software defects once a product has reached the field, any information that can be provided to developers in-process and can give an indication of software defect density is invaluable. If corrective actions can be taken earlier in the software development life cycle to isolate and repair software defects, overall maintenance costs can decrease.

The DevCOP method that we are proposing will help with this problem in several ways. First, after a set of certificates has been created, an overall estimate of defect density can be created based on the V&V weightings using a parametric model. Research has shown that parametric models using software metrics can be an effective method for predicting defect density [10]. We are gathering data from numerous industrial programs to calibrate our method to the general case.

DevCOP also allows developers to manage the effort that is put into V&V in a place where all developers can see what measures have been taken to ensure a piece of code is reliable and to treat it accordingly. The DevCOP method assists developers in identifying and analyzing sections of code that have not yet been certified, or to concentrate their efforts on a particularly critical section of code.

In addition to providing a defect density estimate, DevCOP information can be used to provide a V&V history for particular code segments. Development teams can see what efforts were used to verify the code, even if a different team was working on the system or if poor documentation was available. If the code is found to be error-prone, the certificate information can provide guidance as to what techniques might need to be improved in the organization. During system maintenance, certificates can be referenced to see what types of V&V techniques were performed on a given section of code. If the code is found to be trustworthy, the certificate information with this code could provide evidence that this code is reliable for reuse in future systems.

Another potential use for this V&V information is to build the certificates into the compiled program itself, allowing it to be referenced at runtime by other systems. One possible way of including certificate information with a system is to instrument the code with the certificates, thus storing this V&V effort in a manner that can be reference at runtime. This stored certificate information could prove to be useful information for numerous types of systems, from trust management for personal computers to systems that require load balancing or job distribution. Systems that have information that shows that effort was put in to make the system reliable could receive a greater share of distributed jobs. This potential use of DevCOP is similar to that of security certificates over the Internet [7], which show that effort has been put forth to ensure that the connection or website is secure. The V&V information could also be made available at the time that the system is delivered, to help show the customer that the proper techniques were used to ensure the quality of the product.

7. REFERENCES

[1] Balci, O., "Verification, Validation, and Accreditation of Simulation Models," *Winter Simulation Conference*, 1997, pp. 125-141.

[2] Boehm, B. W., "Building Parametric Models," *International Advanced School of Empirical Software Engineering*, Rome, Italy, September 29, 2003.

[3] Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B., Steece, B., Brown, A. W., Chulani, S., and Abts, C., *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall, 2000.

[4] Dolbec, J. and Shepard, T., "A Component Based Software Reliability Model," *Conference of the Centre for Advanced Studies on C*, 1995.

[5] Fagan, M., "Design & Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182-211, 1979.

[6] International Society of Parametric Analysts, "Parametric Estimating Handbook." Available Online. Online Handbook. http://www.ispa-cost.org/PEIWeb/Third_edition/newbook.htm.

[7] Kent, S., "Evaluating certification authority security," *IEEE Aerospace Conference*, Aspen, CO, March 21-28, 1998, pp. 319-327.

[8] Musa, J., "Theory of Software Reliability and its Applications," *IEEE Transactions on Software Engineering*, pp. 312-327, 1975.

[9] Musa, J., *Software Reliability Engineering*: McGraw-Hill, 1998.

[10] Nagappan, N., "A Software Testing and Reliability Early Warning (STREW) Metric Suite," PhD Dissertation, North Carolina State University, 2005.

[11] Rivers, A. T., Vouk, M.A., "Resource-Constrained Non-Operational Testing of Software," *International Symposium on Software Reliability Engineering*, Paderborn, Germany, 1998, pp. 154-163.

[12] Sherriff, M., Nagappan, N., Williams, L., and Vouk, M. A., "Early Estimation of Defect Density Using an In-Process Haskell Metrics Model," *First International Workshop on Advances in Model-Based Software Testing*, St. Louis, MO, May 15-21, 2005, pp. To appear.

[13] Sherriff, M., Williams, L., "Tool Support For Estimating Software Reliability in Haskell Programs," *Student Paper, IEEE International Symposium on Software Reliability Engineering*, St. Malo, France, 2004, pp. 61-62.

[14] The Programatica Team, "Programatica Tools for Certifiable, Auditable Development of High-Assurance Systems in Haskell," *High Confidence Software and Systems*, Baltimore, MD, 2003.

[15] Williams, L. and Kessler, R., *Pair Programming Illuminated*. Boston: Addison-Wesley, 2002.