DevCOP: A Software Certificate Management System for Eclipse

Mark Sherriff and Laurie Williams
North Carolina State University
{mssherri, lawilli3}@ncsu.edu

Abstract

*During the course of software development, developers will employ several different verification and validation (V&V) practices with their software. However, these efforts might not be recorded or maintained in an effective manner. We have built **D**efect **E**stimation with **V**&V **C**ertificates **o**n **P**rogramming (DevCOP), a software certificate management system. With DevCOP, developers can automatically track and maintain a persistent record of the V&V practices used during development via certificates. With this V&V information, developers and managers can better manage their V&V efforts within a system. Detailed information such as coverage of particular V&V techniques over the system or the amount of V&V performed on a single function can be provided. Developers can also use this V&V information post hoc to see which techniques were more effective at removing defects. In our future work, we are researching a parametric model which utilizes these certificates to estimate defect density as development proceeds.*

1. Introduction

During software development, teams will use several different methods to make a system more reliable [18]. However, the verification[1] and validation[2] (V&V) practices used to make a system reliable might not always be documented effectively, or this documentation may not be maintained properly. This lack of documentation can hinder other developers from knowing what V&V practices have been performed on a given section of code. If developers do not know where V&V has been used, extra time could be spent re-verifying an already thoroughly-verified section of code, or worse, a section of code could go unverified. Further, this information could be used post hoc to see what V&V techniques were used on sections of code that have reported failures from customers. Using this failure information could help developers refine their V&V efforts for future projects.

A development team could benefit from a system that provided a means of V&V *evidence management*. In a software quality context, evidence management is a means of gathering the artifacts and other forms of evidence that a V&V technique was performed to improve V&V documentation efforts [10]. This evidence can take the form of log files, written documentation, information in team management software, or anything else that records V&V effort.

A software certificate management system (SCMS) can support this evidence management. A *software certificate management system* provides an interface and infrastructure to create, maintain, and analyze software certificates [6, 19]. *A certificate* is a record of a V&V practice employed by developers and can be used to support traceability between code and the evidence of the V&V technique used [6, 14].

Our objective is to provide an automated method which allows developers to track and maintain a certificate-based persistent record of the V&V practices used during development and testing. These records could then be leveraged to improve the development process by monitoring V&V system coverage and providing a V&V reference for software maintenance and future projects. To accomplish this objective, we have developed **D**efect **E**stimation with **V**&V **C**ertificates **o**n **P**rogramming (DevCOP). DevCOP is a SCMS which can be use for creating a persistent record of V&V practices as certificates. The DevCOP SCMS will enable us to achieve our longer-term research objective of leveraging certificate information to estimate the defect density of a program.

[1] The IEEE defines verification as "The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."

[2] The IEEE defines validation as "the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements."

We have developed the DevCOP SCMS as a plug-in for the Eclipse[3] integrated development environment (IDE). In this paper, we describe our work in developing and evaluating Version 0.2 of the DevCOP SCMS Eclipse plug-in to support the creation and maintenance of DevCOP certificates and the DevCOP parametric model.

In Section 2, we describe the background of DevCOP and related work. Section 3 provides an overview and demonstration of the DevCOP SCMS plug-in. Section 4 discusses our limitations. Sections 5 and 6 describe our future work and a summary of what we have accomplished to date.

2. Background and Related Work

In this section, we will discuss the relevant background work and methodologies used during our research, including V&V techniques; SCMSs in general, and one particular SCMS, Programatica, which particularly influenced our work.

2.1 Types of V&V 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/or 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 [18]. 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 [7] 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 [20] 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. 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.

Balci categorized V&V techniques with regard to the technique's methodology for detecting defects [1]. Balci's categorization of V&V techniques includes:

- Manual – includes all manual checking, such as pair programming [20] and code inspections [7];
- 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 black-box testing;
- Formal – includes all strictly mathematical forms of checking, such as lambda calculus and formal proofs [18].

2.2 Software Certificate Management Systems

Different V&V techniques produce various kinds of evidence of their execution, such as logs from test cases, written records from code inspections, or systems that can record pair programming efforts. The evidence from these V&V techniques could be stored manually in various forms of documentation. However, manually recording static documentation takes time away from development and produces documents that are often not maintained after their initial creation. If these V&V documents are not maintained, they become obsolete and the effort is wasted.

Software certificates provide a valuable resource for developers to gather V&V information, in some cases automatically, in a single format and to also increase traceability between V&V techniques and sections of code. This V&V information can be used for maintenance purposes, for analysis of the effectiveness of certain V&V practices, for future reference in reused code, or for defect density estimation purposes. Information about who performed various V&V techniques could be useful for software maintainers as they have a better idea who to talk to if they have issues with a particular system. Developers could learn about the effectiveness of their V&V practices if they compare defect reports to the coverage of particular V&V techniques. If numerous defects are found in sections of code that have all been checked with a certain V&V technique, developers might adjust the use of that technique or spend time refining it. System modules are also often reused in later projects within an organization. Having an accurate record of the V&V practices used on that code could help speed development, as teams might not spend extra time verifying a previously-verified section of code. In our research, we are developing a method of estimating defect density in a system based on its record of V&V techniques.

While the creation of software certificates allows for the collation of V&V information in a single place,

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

these certificates must be maintained so that they accurately reflect the current code base. A SCMS provides a range of services, including automatically creating and maintaining certificates, enabling the browsing of certificates in a system, and checking the validity of certificates (e.g. automatically invalidating a certificate upon a code change) [6, 19]. The goal of a SCMS is to automate the management of certificate information so that minimal overhead is added to the development process. Research is being conducted by various groups in how SCMSs can be used in the development process [6, 8, 10, 19].

2.3 Programatica

Programatica [10, 18] is a SCMS that has been in development since 2003. This system is the inspiration for the work mentioned in the previous section and for our DevCOP work. The Programatica team at the Oregon Graduate Institute at the Oregon Health and Science University (OGI) and at Portland State University (PSU) is working on a method for high-assurance software development for the Haskell programming language [10, 18]. The goals of the Programatica team are to allow users to capture evidence of V&V and to manage this evidence to help guide future development efforts [10]. The Programatica tool is built on the concept that specified properties could be placed in the source code itself to show that certain pieces of code have been verified or validated through a particular V&V technique. These properties can be derived from several different sources, such as expert opinion, test cases, or external theorem provers. These specified properties become certificates, linking a validated property as evidence of high-assurance with a piece of code.

Programatica allows various external tools to plug-in to its certificate management module so that Programatica can leverage the V&V evidence provided by these tools. For example, Programatica gathers V&V evidence from an external testing framework called QuickCheck [4] and a theorem prover called Alfa [10]. Developers can write and then certify code as it becomes complete with these external tools.

Managers can decide how much of the code base needs to be certified at any given time, slowly increasing this number as the system nears completion [10, 18]. Programatica uses this idea of gradually increasing the number of certificates since code is usually more in flux at the beginning of a project, and a developer's time should not be spent recertifying code that could change soon after recertification.

The DevCOP SCMS plug-in builds on the ideas initiated in the Programatica project, but expands on them in various ways. The main difference is that DevCOP is written for Java rather than Haskell. The Programatica tool is used in conjunction with a developer's programming environment of choice. DevCOP, however, is directly integrated into the Eclipse IDE. The plug-in can also interact directly with the programming tools that the developer uses, such as test case coverage tools like jcoverage[4]. DevCOP also adds basic reporting tools, such as views that provide information to the developer on V&V certificate coverage.

3. The DevCOP SCMS Eclipse Plug-in

We have created the DevCOP Version 0.2 SCMS Eclipse plug-in[5] to handle the creation and management of V&V certificates during the development process [12, 17]. In this section, we will describe early versions of the tool, our implementation of software certificates, and the features of the plug-in.

3.1 Early Versions

DevCOP Version 0.1 SCMS Eclipse plug-in was released in Spring 2005 as a beta version to several industrial Java development teams for evaluation. Figure 1 shows a screenshot of the plug-in. This screenshot demonstrates how the DevCOP SCMS is integrated into Eclipse. The background of the screenshot shows the Eclipse Java Editor and the DevCOP Certificate Browser view, along with the Certificate Editing dialog. These features of DevCOP will be explained further in later sections of this paper.

Version 0.1 focused on recording certificates that normally do not produce artifacts that are stored with the code. These certificates included such as manual techniques like code inspections and pair programming, and not automatic or programmatic V&V, such as unit testing. Programmers could select one or more functions for certification through the Eclipse Package Explorer and the type of V&V technique used (i.e. code inspection, pair programming, bug fix, etc.). The certificate information was then stored in an XML document that was saved in the project's workspace.

Anecdotal reports from the teams indicated that the initial version of the plug-in did not contain enough functionality to warrant inclusion into their development cycle. There were also concerns about problems that could arise from distributed code development using an XML storage format and loose integration with the Eclipse IDE. Further, the

[4] Jcoverage can be found at http://www.jcoverage.com/.
[5] The plug-in is available at http://agile.csc.ncsu.edu/devcop/.

developers indicated that detailed metrics about the coverage of the different V&V techniques (e.g. how much of the code was pair programmed vs. solo programmed) would be useful in their development efforts.

To address the concerns of our test Java developers, we redesigned and enhanced the DevCOP SCMS plug-in and have released the beta version of DevCOP Version 0.2 SCMS Eclipse plug-in [16, 17]. The architecture and methodology changed significantly in this release to make the system more viable for multiple-developer projects and to include external V&V tools.

3.2 DevCOP Certificates

Based on the Programatica team's work [18], we decided that DevCOP certificates would be created for functions (or methods), as opposed to classes or individual lines of code. This decision was made because a certificate for every line of code could be overly difficult to manage and certificates for classes only might not provide enough information to developers. Each certificate contains the following:

- identifying information for the function it is associated with including its name, signature, class, and file location;
- identifying information for the developer that created it;
- the type of V&V technique used; and
- a hash of the function's abstract syntax tree (AST).

The hash of the function's AST is stored to ensure that a certificate is valid at any given time. Eclipse's Java Development Toolkit enables the DevCOP SCMS plug-in to gather a textual representation of a function's AST. The DevCOP SCMS then performs a MD5 hash on this textual AST and stores it within a certificate. A certificate is considered *valid* if, and only if, a certificate's AST hash matches a hash of the AST of the current source code. This indicates that if a function is modified after it has been certified, that certification is no longer valid. If the code is changed back to its previous state, the certificate becomes valid again. Using a hash of a function's AST allows the DevCOP SCMS to ignore insignificant changes, such as code formatting or comments. A certificate only becomes invalid if the change in the code is significant

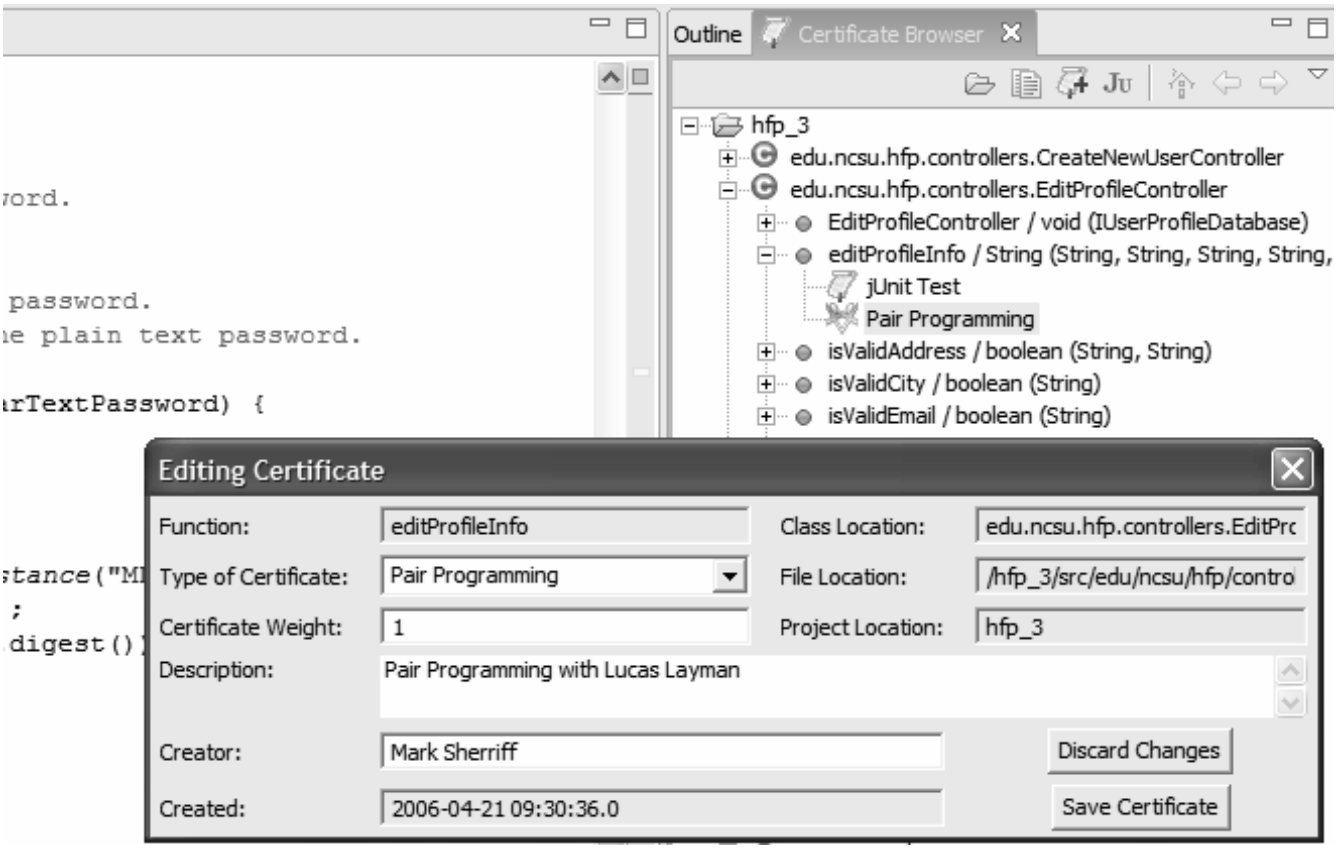

Figure 1. DevCOP SCMS Screenshot

enough to change the Java AST. If the developer determines later that the change did not affect the certification, he or she select that certificate and revalidate it, which replaces the current AST hash within the certificate with the AST hash of the changed function. The certificate is then tagged as being recertified and can be examined later to determine if the recertification was correct

3.3 DevCOP Eclipse Views

The DevCOP SCMS Eclipse plug-in provides several different options for recording and managing V&V information within Eclipse. The three Eclipse views include the Certificate Browser view, the Certificate Coverage view, and the Certificate Weighting view. We have also provided some initial certificate recording mechanisms and an interface to jcoverage, an external test case coverage tool. Figure 2 shows the design of the plug-in.

The certificate manager module serves as the central point to the system and controls the interface to the database which stores the certificates, replacing the XML functionality in the DevCOP Version 0.1. This module provides data for the three display components, the Certificate Browser view, the Certificate Coverage view, and the Certificate Weighting view. The certificate manager accepts and processes new certificates from the certificate creation interface. We have also integrated the certificate manager module into the Eclipse IDE such that it can update certificates appropriately when the Eclipse refactoring tool is used.

DevCOP presents V&V information to the developer within the Eclipse IDE. The main view is the Certificate Browser, which allows developers to go through a list of certificates, sorting by class or by type. A picture of the Certificate Browser is shown on the right hand side of Figure 1. The goal of this view is to provide a simple tree structure to quickly find certain certificates to review or edit its descriptive information, such as general comments about the certificate. The Certificate Browser also allows developers to revalidate certificates that may still be valid, but are marked invalid due to a change in the code, as described in the previous section.

The Certificate Coverage view provides developers with a tabular representation of the V&V coverage in their system, separated by the different V&V techniques. A screenshot of this view can be found in Figure 3. This view calculates basic system metrics such as lines of code and number of methods and then determines which methods are covered by V&V techniques. Developers can see overall V&V coverage by function, class, or system, or for each particular V&V type. Developers and managers can use this information as a quick reference to get an overall picture of how much of the system their V&V techniques are covering. This coverage information can aid developers by isolating certain modules that might not have been covered yet by any V&V technique. The view also provides metrics for those teams that want to track V&V usage, such as Extreme Programming [2] teams that want to see how much of the code based was pair programmed versus solo programmed.

As Balci described in his work, V&V techniques can be put into different categories based on their methodology. We are adding this concept into DevCOP by adding a weighting component to the different V&V certificate types. Weights will be determined via a regression equation on historical data and will be used in our future work in predicting

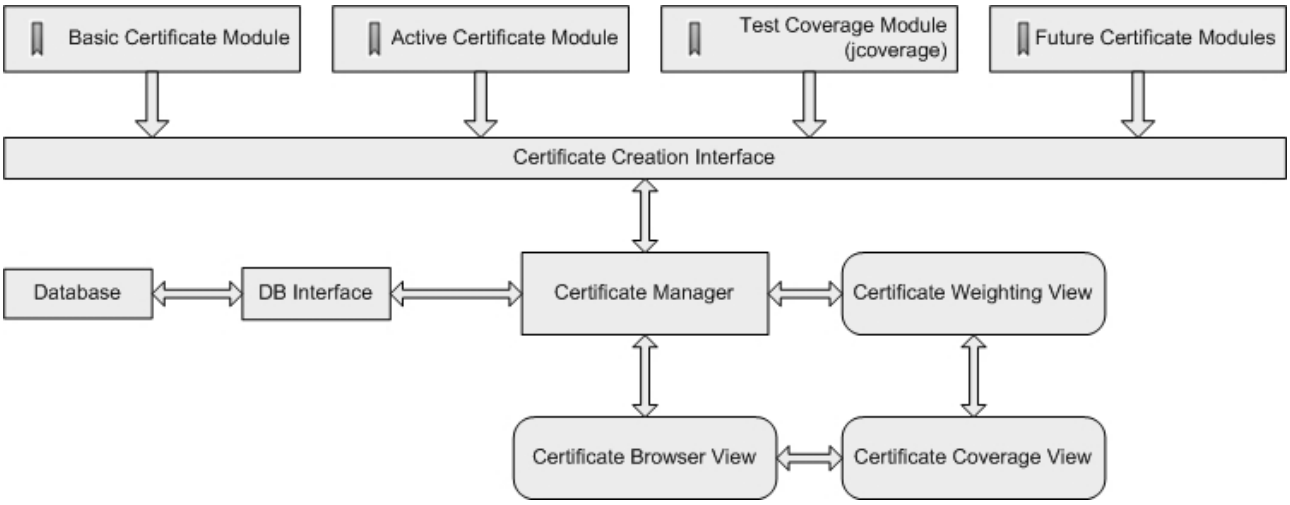

Figure 2. DevCOP SCMS Plug-in Design

Certificate Type	Lines of Code	Covered Lines	% Covered	Methods	Covered Methods	% Covered
Pair Programming	3335.0	426.0	13%	569.0	97.0	17%
DevCOPPlugin	217.0	66.0	30%	21.0	11.0	52%
DevCOPCertificate	60.0	12.0	20%	29.0	12.0	41%
DBInterface	368.0	0.0	0%	12.0	0.0	0%
CompareObject	43.0	43.0	100%	18.0	18.0	100%
JavaMethodDifferencer	37.0	0.0	0%	3.0	1.0	33%
JavaElementListener	55.0	0.0	0%	3.0	0.0	0%
CompareAction	7.0	7.0	100%	3.0	3.0	100%
GenericMetricVisitor	10.0	0.0	0%	5.0	2.0	40%
MetricParent	22.0	3.0	14%	13.0	3.0	23%
Congress	184.0	0.0	0%	4.0	0.0	0%
Counter	18.0	1.0	6%	2.0	1.0	50%
DevCOPCoverage	**309.0**	**270.0**	87%	**14.0**	**11.0**	**79%**

Figure 3. Images from the Certificate Coverage view

software defect density. Currently, we are using default weights in DevCOP, which will be replaced in our future empirical work. Cumulative weights can be viewed in the Certificate Weighting view, a graphical view in the DevCOP SCMS, as shown in Figure 4. The graph in this view shows the sum of the weights of the certificates for the given functions and classes in the system. When a user clicks on any function in the system, either in the Outline view or Package Explorer view, the Certificate Weighting graph is shown for that function. This weighting graph provides a simple view of not only what V&V techniques have been used on this function, but their relative effectiveness within the system based on their regression constant as determined by historical data. The weighting graph can also provide similar information at the class and system levels. This weighting information can provide an overall snapshot of what functions have been adequately covered and which have not. Figure 4.a shows a weighting graph on a single function that has two certificates. Figure 4.b shows a weighting graph on a portion of the system, organized by class.

3.4 DevCOP Certificate Creation Methods

DevCOP includes a basic certificate creation interface that can be utilized by various external V&V tools to create certificates. We have provided two

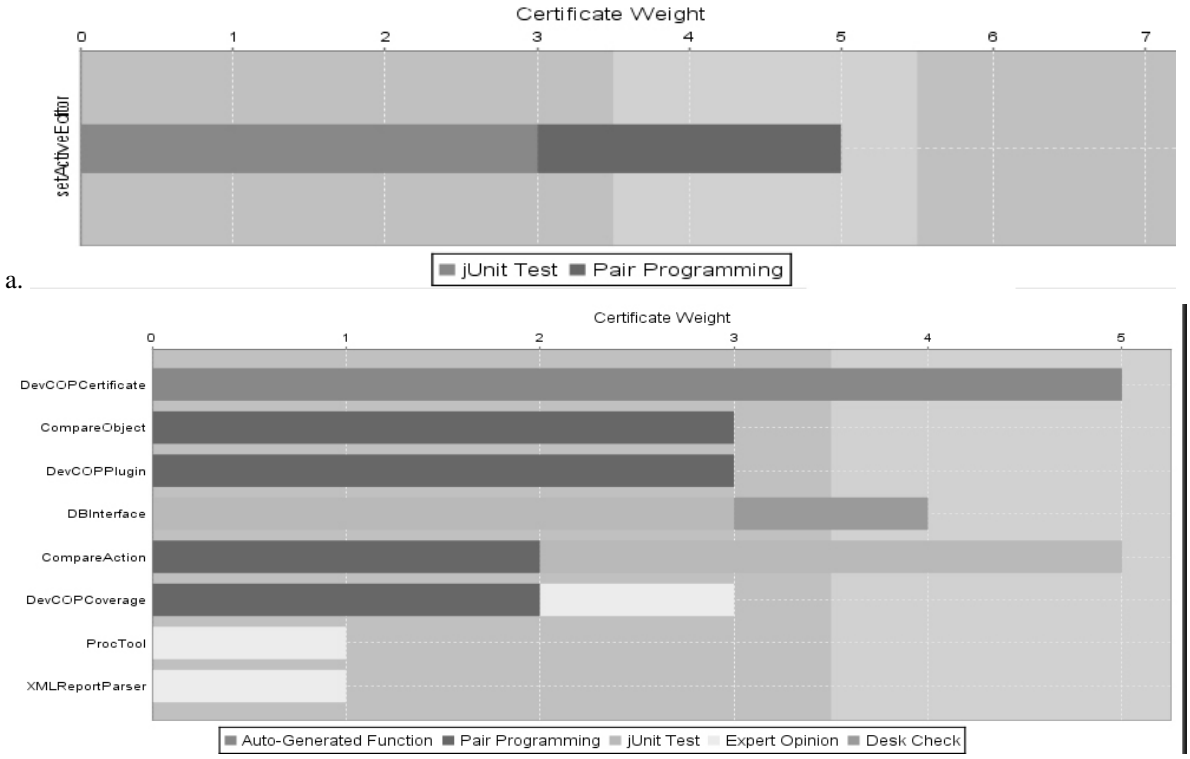

Figure 4.a (top) Weighting for one function; 4.b (bottom) Weighting for multiple classes in the system

internal certificate creation modules (the basic certificate module and the active certificate module) and one external tool interface (the test case coverage module utilizing jcoverage). Table 1 provides a list of the certificates that can currently be created by DevCOP.

The basic certificate module allows for the manual creation of certificates on one or more functions using Eclipse's graphical interface. Developers can right-click on any function (or group of selected functions) in Eclipse's Package Explorer or Outline view and choose to add a certificate to that function from its context menu. The user can then add more detailed information about what V&V technique was used to authorize the creation of this certificate. This basic method of adding certificates allows developers to target individual functions at any time for V&V techniques that do not have external tools or an automated certificate creation method, such as a references to expert opinion or desk checking one's own code.

Developers, however, may wish to add multiple certificates at once, across several files or even projects, such as during a long pair programming session or code review. Remembering or writing down every "touched" function during these sessions could be difficult and error prone and would make certificate creation cumbersome and labor-intensive.

To facilitate the creation of certificates using V&V techniques like pair programming, we have created what we call active certificates. An *active certificate* is a means by which Eclipse will automatically identify changed code during a programming session to be certified by the developer. For example, if two programmers were about to start pair programming on a piece of code, they would click the active certificate button before they began. Eclipse would then actively record non-trivial changes to the system (i.e. changes to the abstract syntax tree of the code, not commenting or formatting changes) and will present the affected functions to the developers for certification at the end of the pair programming session. The concept of active certificates can extend to several different types of V&V activity, such as pair programming or bug fixes. Active certificates allow developers to write or modify code normally, without increasing their work overhead.

While the basic and active certificate modules allow for the collection of most manual V&V techniques, we want to leverage the numerous automated V&V tools that are currently available and are being used actively in industry. Each of these tools aids developers in different ways, looking at various aspects of the code base.

The first tool we have created an interface for is jcoverage. Jcoverage, licensed under the GNU General Public License, is an extension to the Apache Ant build tool. This tool instruments Java code to allow for the collection of test case coverage in a system. We have integrated the output from the jcoverage tool to allow DevCOP to initiate jcoverage within the IDE and then to collect metrics regarding test case coverage. Currently, jcoverage uses JUnit test cases to compute test case coverage, but we are expanding this feature to allow developers to specify particular test case files. DevCOP will automatically create a test case coverage certificate for each function whose test case coverage is above a particular threshold specified by the development team. The test case coverage percentage threshold can be changed in the DevCOP preferences, and is recorded with each generated certificate. It is important to note that currently DevCOP does not support the accumulation of coverage as is done with asynchronous testing such as block box function or system testing. We are adding asynchronous testing into the next version of DevCOP. We will add more tool interfaces to DevCOP as it continues in development, such as the ESC/Java2 static analysis

Table 1. Currently Available Certificate Types

Certificate Type	**Category**	**DevCOP Certificate Creation Method**
Code Inspection	Manual	In Eclipse, right click on any function or group of functions and click Add Certificate
Code Review	Manual	
Desk Check / Walk-through	Manual	
Expert Opinion	Manual	
Pair Programming	Manual – Active Certificate	Click *Begin Active Certification* in the Certificate Browser and Eclipse will record all non-trivial code changes for certification at the end of the certification session
Bug Fixes	Manual – Active Certificate	
JUnit test case coverage	Dynamic	Click the JUnit icon in the Certificate Browser to launch jcoverage, which will in turn run the system's test suite

tool [5]. We are also developing an extensible interface for other Eclipse plug-ins so that developers can create a certificate creation interface for their own V&V tools.

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-[11] based V&V. 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 goals 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 limitation is the granularity of certificates. Based on the Programatica Team's work [18] it was decided that methods would be the proper level of granularity for certificates. Finally, programmers can manually add or change certificates within the system, so the system is not completely objective and/or audit-safe. If a developer recertifies a certificate, functionality needs to be in place to force the user to justify changes to a certificate.

5. Future Work: The DevCOP Parametric Model

DevCOP Version 1.0 will include defect density estimation via a parametric model [13] which utilizes the certificates stored in the DevCOP SCMS [12, 15]. 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 [9]. The goal of a parametric model in software engineering is to provide an estimated answer to a software development question earlier in the development lifecycle.

We have worked together with the Center for Software Engineering at the University of Southern California [3] to create a parametric modeling process specifically for software engineering research [13]. This process, illustrated in Figure 5, shows the steps that can be followed to create an effective parametric model. More information on the individual steps can be found in [13].

We are integrating our estimation directly into the development cycle using the DevCOP SCMS plug-in so that developers may take corrective measures and perform more V&V earlier in the development lifecycle. The goal of the model is to provide an estimate of defect density based on V&V certificates and the coverage of each certificate type. We anticipate that a model would need to be developed for each programming language we would study.

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 weight 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 validate our parametric model and significance weight coefficients, we will perform a causal analysis with our industry partners on Java projects currently in development. Once four to six months of field failure information is available, we can compute an actual defect density and compare it with our estimate. We can also perform causal analysis to provide more information about the efficacy of certain techniques under particular circumstances.

6. Summary

We have created and are currently evaluating a SCMS plug-in for Eclipse. The DevCOP SCMS plug-in allows developers to easily record their V&V activities within the development environment with minimal increase in overhead. By utilizing a SCMS, V&V information is stored in a single format in a single location and is maintained automatically. Using a SCMS can significantly reduce the time required to record and maintain V&V information, while also providing reporting tools to leverage this information during the development cycle.

The DevCOP plug-in currently allows for the creation and management of manual and JUnit test case coverage V&V certificates and provides developers with reporting tools to evaluate their V&V efforts. The integration into the Eclipse IDE enables developers to leverage the mechanisms already in Eclipse to automatically create certificates through our active certificate method. There is also a basic mechanism for incorporating other automated V&V tools, such as code coverage tools like jcoverage, which will be improved in future versions.

Three Eclipse views are provided for developers to examine their V&V efforts. These views allow developers to manage certificates, view V&V technique coverage over the system, and to see what

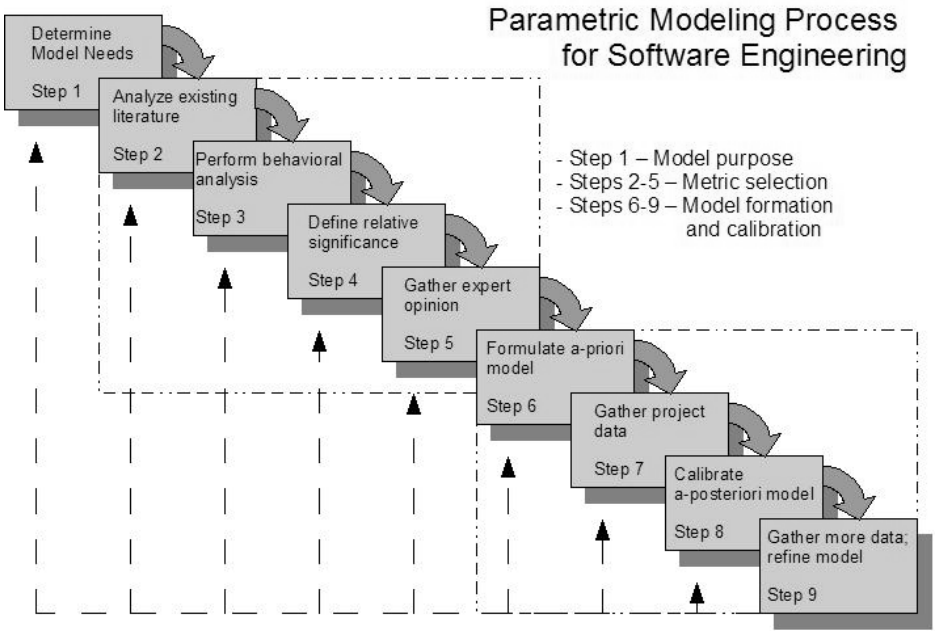

Figure 5. Parametric Modeling Process for Software Engineering

V&V techniques have been performed on single functions or classes. The V&V information could be used in-process to show what areas of the system might need to be verified more, or the information could be used post hoc to analyze the effectiveness of V&V techniques or for software maintenance. We are also developing a method for a development team to estimate software defect density in-process using this V&V information. We will continue our work to improve the plug-in based on developer suggestions and to gather data to validate the DevCOP parametric model.

7. Acknowledgements

We wish to give our most sincere thanks to Dr. Mark Jones at Portland State University and rest of the Programatica team for their input on the various parts of this work. This work was funded by the National Science Foundation.

8. References

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

[2] Beck, K., *Extreme Programming Explained: Embrace Change*, Second ed. Reading, Mass.: Addison-Wesley, 2005.

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

[4] Classen, K. and Hughes, J., "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," International Conference on Functional Programming, Montreal, Canada, Sept. 18-20, 2000, pp. 268-279.

[5] Cok, D. and Kiniry, J., "ESC/Java2: Uniting ESC/Java and JML - Progress and issues in building and using ESC/Java2," Construction and Analysis of Safe, Secure and Interoperable Smart devices Workshop, Marseille, France, March 10-13, 2004, pp. 108-128.

[6] Denney, E. and Fischer, B., "Software Certificate and Software Certificate Management Systems," Workshop on Software Certificate Management, Long Beach, CA, Nov 8, 2005, pp. 1-6.

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

[8] Hutter, D., "Software Certification Management: How Can Formal Methods Help?" Workshop on Software Certificate Management, Long Beach, CA, Nov 8, 2005, pp. 47-50.

[9] International Society of Parametric Analysts, "Parametric Estimating Handbook," Department of Defense, Online Handbook, April 16, 2005. Available online: http://www.ispa-cost.org/PEIWeb/Third_edition/newbook.htm

[10] Jones, M., "Evidence Management in Programatica," Workshop on Software Certificate Management, Palm Beach, California, 2005.

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

[12] Sherriff, M., "Using Verification and Validation Certificates to Estimate Software Defect Density," Doctoral Symposium, Foundations of Software Engineering, Lisbon, Portugal, September 6, 2005, 2005.
[13] Sherriff, M., Boehm, B. W., Williams, L., and Nagappan, N., "An Empirical Process for Building and Validating Software Engineering Parametric Models," North Carolina State Univeristy CSC-TR-2005-45, October 19 2005.
[14] Sherriff, M. and Williams, L., "A Method for Verification and Validation Certificate Management in Eclipse," Workshop on Software Certificate Management, Long Beach, CA, Nov 8, 2005, pp. 19-22.
[15] Sherriff, M. and Williams, L., "Certification of Code During Development to Provide an Estimate of Defect Density," Fast Abstract, International Symposium on Software Reliability Engineering, Chicago, IL, Nov 8, 2005, pp. 447-448.
[16] Sherriff, M. and Williams, L., "Defect Density Estimation Through Verification and Validation," The 6th Annual High Confidence Software and Systems Conference, Lithicum Heights, MD, April 17-19, 2006, pp. 111-117.
[17] 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.
[18] The Programatica Team, "Programatica Tools for Certifiable, Auditable Development of High-Assurance Systems in Haskell," High Confidence Software and Systems, Baltimore, MD, 2003.
[19] Whalen, M., "Certificate Management: A Practitioner's Perspective," Workshop on Software Certificate Management, Long Beach, CA, Nov 8, 2005, pp. 23-26.
[20] Williams, L. and Kessler, R., *Pair Programming Illuminated*. Boston: Addison-Wesley, 2002.