Dione: An Integrated Measurement and Defect Prediction Solution

Conference or Workshop Item

Dione: An Integrated Measurement and Defect Prediction Solution

Bora Caglayan
Department of Computer Engineering
Bogazici University
Turkey
bora.caglayan@boun.edu.tr

Ayse Tosun Misirli
Department of Information Processing Science, University of Oulu, Finland
ayse.tosunmisirli@oulu.fi

Gul Calikli
Department of Computer Engineering, Bogazici University, Turkey
gul.calikli@boun.edu.tr

Ayse Bener
Ted Rogers School of Information Technology Management,
Ryerson University, Toronto, CA
ayse.bener@ryerson.ca

Turgay Aytac
Prescience Inc., USA
taytac@scidesktop.org

Burak Turhan
Department of Information Processing Science, University of Oulu, Finland
burak.turhan@ oulu.fi

ABSTRACT
We present an integrated measurement and defect prediction tool: *Dione*. Our tool enables organizations to measure, monitor, and control product quality through learning based defect prediction. Similar existing tools either provide data collection and analytics, or work just as a prediction engine. Therefore, companies need to deal with multiple tools with incompatible interfaces in order to deploy a complete measurement and prediction solution. Dione provides a fully integrated solution where data extraction, defect prediction and reporting steps fit seamlessly. In this paper, we present the major functionality and architectural elements of *Dione* followed by an overview of our demonstration.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – productivity, software quality assurance.

General Terms
Management, Measurement, Economics, Reliability.

Keywords
Software tool, measurement, software defect prediction.

1. INTRODUCTION
In recent years, different approaches to implement defect prediction models have become increasingly popular [3]. Using these approaches, practitioners seek to find efficient and cost-effective solutions to solve software engineering problems such as software measurement and analysis, software quality estimation and efficient resource allocation [1,2].

Existing measurement and decision support models often depend on expert judgment based on ad-hoc procedures. Moreover, they handle only a portion of the key needs which consist of software measurement, analysis and defect prediction. Companies are forced to use separate tools with incompatible APIs in order to implement an integrated solution covering measurement, analysis and decision support functionalities. Such an integrated solution is too cumbersome to implement especially for small and medium-sized enterprises (SMEs). An integrated solution that addresses the measurement, analysis and decision making requirements of SMEs is necessary to make them competitive in the market.

In this paper, we propose an integrated measurement and defect prediction tool: Dione. Our main motivation behind such an initiative was to produce a major infrastructure in order to help software managers in the development of measurable, consistent and reliable software products. Dione automatically collects data from version control systems (CVS, SVN, Git, Mercurial, Clearcase) and bug repositories (Bugzilla, Jira), which are popularly used in the software industry, and it extracts software metrics. Dione also uses the extracted data to predict the defect-proneness of existing software artifacts using machine learning techniques.

SIGSOFT'12/FSE-20, November 11 - 16 2012, Cary, North Carolina, USA

2. DIONE AT A GLANCE
Dione is implemented in Java as a web application. The major functions of Dione, which can connect to company's source code/issue management systems, are as follows:

- Building a measurement repository that contains product and process metrics (e.g. [2,3]) as well as information about defected software components (e.g. files, classes, methods)
- Analyzing trends in metrics and issues using chart and report configurations
- Constructing and calibrating customized a defect prediction models to predict defect proneness of a software product's version or release.

2.1 Main Functionality
The main benefits of Dione over similar tools is integrating data extraction, reporting and defect prediction modules for a complete decision support cycle and offering a minimum configuration solution. In this section, we will explain the modules of Dione in more detail.

Automatic Metric Extraction: Dione uses smart client technology in order to connect to software development artefacts (e.g. source code repositories, version control systems and issue management systems) and to automatically extract product and in-process metrics. Product metrics are static code metrics and a list of static code metrics extracted by Dione can be found in [1]. In-process metrics capture information about changed lines of code, committers and their commit frequencies. Supported languages for metric extraction are Java, C, C++, PL/SQL, and BPEL for Oracle. By applying a pattern matching heuristic, Dione can also detect and label defective parts of a software product at a pre-defined granularity level (e.g. package, file, class or method). An example heuristic is based on mining commit messages to extract issue IDs or a list of words (pre-defined with the software team) indicating a bug fix on the source code.

Manual and Batch Recording of Metrics: On the server side, Dione stores the metrics gathered by smart clients in a Relational Database Management System (RDMS). We do not restrict organizations to use a specific RDMS since Dione can easily connect to popular databases, such as Oracle, DB/2, MS SQL Server, MySQL, PostGreSQL. Also, organizational data (projects, processes, organizational teams, etc.) can be entered using web-based forms manually.

Defect Prediction: The defect prediction module uses machine learning algorithms (e.g. Naive Bayes, Bayesian Networks, Decision Tree, Logistic Regression, Neural Networks, Cascading classifiers). The system learns from historical metrics and defect data and predicts defect proneness of current version (or release) of a software. This learning-based approach has advantages over rule-based models, since a) it learns from historical data without human intervention, and b) the prediction performance of learning based models reach to 80% defect detection rates, while significantly reducing the inspection efforts for finding defects [2].

In the case of the absence of local data, we used the approach proposed by Turhan et al. [3] to train the model with cross project datasets stored in the server side of Dione. Results in [3] show that cross-project data can serve as meaningful proxy in the early stages of a metrics program. Dione also provides a web UI that allows users to calibrate their local defect prediction model with custom algorithms, pre-processing techniques and heuristics before running their model. An example usage can be seen in Section 3.4.

Web-Based Reporting: This module presents the output of the prediction module to the users. The reports are custom tailored to different stakeholders of the organization. For example, code level predictions (as the list of defective/defect-free source files or the probability of defect proneness of files) are presented to developers in order to direct them to the critical parts of the code for additional quality assessments, such as unit tests and inspections. Diagrams summarizing defect density and active developers working on defect prone modules are presented to project managers in order to help them manage and optimize testing resources.

2.2 Architecture

Dione utilizes all advantages of cloud computing which can be defined as the delivery of computing and storage capacity as a service to a scalable number of end-recipients. Therefore, customer maintenance costs are minimal and also the storage and computing resources of Dione can be used by multiple customers remotely. Data mining tasks such as defect prediction, which are computationally intensive, are implemented in the form of atomic components so that they can be executed in parallel and can be scaled out to several servers. Computationally intensive tasks can be scheduled and monitored through a web-based environment and these tasks utilize multiple server/processor resources. The complete system architecture for Dione is shown in the architecture diagram in Figure 2.

The web client is developed in Java by using Google Web Toolkit (GWT) in addition to Javascript libraries such as jQuery and HighCharts. Input forms and reporting modules were developed in Java. For cross-browser compatibility, these modules compile using GWT. Visual templates for chart types were developed in Javascript using HighCharts library. Numerical computations and intensive parts such as the machine learning algorithms were implemented using Apache JMath library.

One of the most important innovations in the architecture of Dione is the smart clients which are small Java programs that can be executed directly from the Dione web interface. Smart clients gather metrics from the software artifacts such as source code repositories and issue management systems, and they send these data to the Dione server. In order to enhance functionalities on the client-side, one can download smart clients for log-keeping and process monitoring from the server side. Integration with other software quality applications is supported through web services.

Smart clients retrieve configuration data that they need from the server side. Dione also satisfies privacy constraints, since smart clients do not store any data or configuration information such as passwords and user names permanently. Moreover, data transferred between server and client sides are encrypted, as well as the data stored temporarily on the client side. Smart clients are updated automatically by changing the version of the smart client on the server side.

Dione offers two solutions for different organizational needs, namely fully centralized and on-site. In fully centralized solution, all data are stored on the central Dione server. This version of Dione collects data from multiple companies into one repository. Data privacy is ensured by organizational segmentation. In the on-site solution, the server is deployed within the company network.

2.3 Dione Availability

Access details for the online trial version of Dione can be requested by contacting info@softlabint.com.

3. REFERENCES

[1] Kocaguneli E., Tosun, A., Bener, A., Turhan, B., Caglayan, B., Prest: An Intelligent Metric Extraction, Analysis and Defect Prediction Tool, Proceeding of the 21st Int. Conference on Software Engineering and Knowledge Engineering, July 1-3, 2009, Boston, USA.

[2] Tosun, A., Bener, A., Turhan, B., Menzies, T., Practical Considerations in Deploying Statistical Methods for Defect Prediction: A Case Study within the Turkish Telecommunication Industry, Inf. and Software Technology, 52(11): 1242-1257, 2010.

[3] Turhan, B., Menzies, T., Bener, A., Distefano, J., On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering, 14 (5): 540-578, 2009.

[4] Eclipse Project Web Page, http://www.eclipse.org/.

APPENDIX: Dione Demo

We will use the well-known open source software Eclipse to show the metric extraction, metric analysis and defect prediction features of Dione in the demo. To accomplish this, we setup a server with the project data of Eclipse. All operations will be done by users through web browsers to access the Dione server during the demo. Demo steps can be tried by multiple users concurrently.

Demo Setup Details

Dione hardware setup during the demo includes two networked PCs with server capabilities. All the required hardware for the demo will be brought by the demo team. Deployment schema of Dione can be seen in Figure 1.

The first PC act as the client layer. It will also store the cached source code repository of the Eclipse project. The second PC hosts the server side components of Dione. The program is written in Java and Apache Geronimo Application Server is used to serve both smart clients and web pages. The data is hosted on a Mysql database on the second PC. Under normal conditions, demo of Dione will take between 15 to 20 minutes.

As the first step, all relevant Eclipse project data imported to the database so that a client-deployed implementation of Dione can access required data and use them in all operations. Internet access will be required to access Dione server during demo. Public access to Dione will be available throughout the demo.

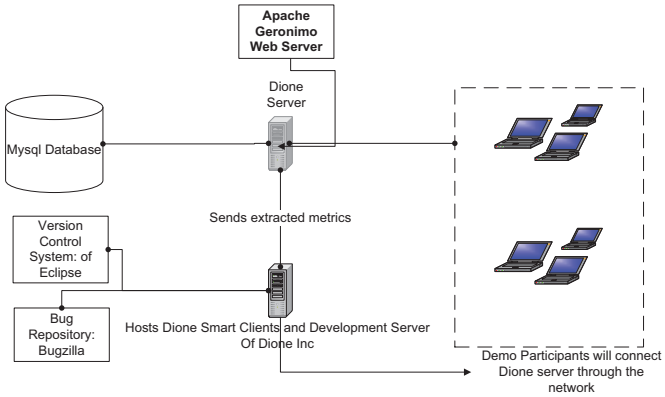

Figure 1 Deployment details of Dione

Demo Organization Details

The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate both an open source community and an ecosystem of complementary products and services [4].

Eclipse Project

Eclipse project is a widely used multi-language software development platform written mainly in java language. It was initially derived from IBM VisualAge IDE. The project was made open-source in November 2001 and it was licensed initially under creative-commons license and later Eclipse Public License. Both of these licenses are compatible with FSF standards for open source licenses. During the history span of Eclipse project that we examined, the core committers to the project were IBM employees.

In the Demo we will use Dione to extract and import the static code and churn metrics of the Eclipse Project during the development between January 2003 and June 2006. We will use the extracted metrics to show the analysis capabilities of Dione and predict the post release defects for Eclipse versions 3.0, 3.1 and 3.2 using Dione and show the prediction output.

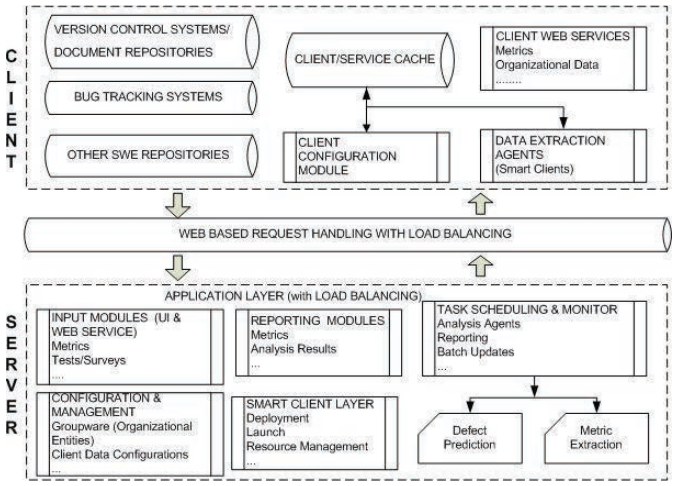

Figure 2 Architecture Document of Dione

Steps of the Demo

Data Import

Easy integration with existing project data is a major strength of Dione compared to other solutions. Data import in Dione can either be done automatically through the smart clients or manually using xml files. During the demo we will show a data import operation using both available methods. Before the demo, we will predefine the relevant organizational data, such as organization, team, roles, project and version, using our web application in order to save time.

For importing the change sets of Eclipse and entities related to the change sets we will use XML files for the bulk of the data. Afterwards we will show the live data transfer capability of the smart clients. Smart clients will be configured with the Eclipse project connection parameters and run from the web interface of Dione.

Imported files, classes, methods and metrics extracted from these software modules via Prest can be seen and edited from Dione's web application (Figure 3). Figure 3 shows the granularity, source (e.g. method name), metric code and metric values imported from a

project. We will show the imported changeset and metric data of Eclipse during the demo.

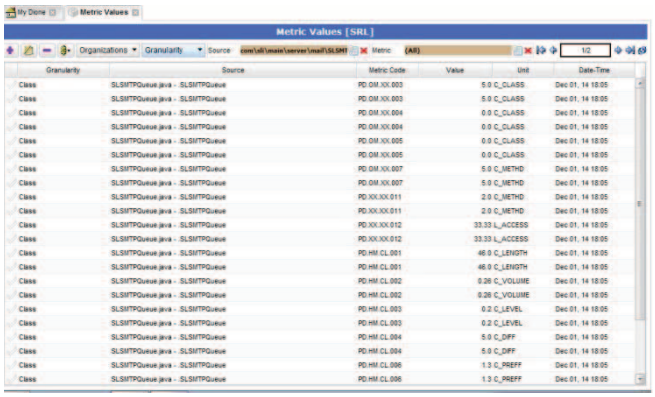

Figure 2 Metric Viewer In Dione

Reporting

Metrics are the building blocks of the charting and reporting system of Dione. Changes in key indicators in every software entity can be tracked over time or change periods. Past results of the intelligent modules can be tracked through summary results. If subscribed, reports and charts are sent by e-mail in time intervals.

In addition, Dione features a mail alert mechanism that warns recipients about potential problematic situations. In Figure 4, change trend of lines of code of a file in the software product over revision history is shown as an example.

Derived metrics that aggregate or combine base metrics can be defined within Dione for reporting purposes. Dione can be scheduled to derive and cache the derived metrics in order to save time during reporting.

In the demo, we will show the interesting trends observed during Eclipse development: 1) Calculation of a derived metric: cyclomatic complexity per month, 2) LOC size of the project over time, 3) Defect count and defect density per month, 4) Average cyclomatic complexity per month.

Figure 4 Metric Analysis Screen For Eclipse Project

Defect Prediction

This component is one of the four intelligent modules built into Dione. It has a configuration and a prediction step. In configuration, historical data, test data and algorithms that will be used during defect prediction are selected by the user.

Furthermore, periods of defect prediction runs (i.e. every day/week/month) can be defined so that Dione periodically produces reports including predictions about defect-proneness of software modules and metrics that are most significant indicators of defects. It is also possible to run these configurations manually and observe their progress or errors (if exists).

In the configuration step, Dione provides to the user the ability to define custom prioritization in addition to default labels assigned in prediction step. More specifically, it is possible to define alert levels, such as LOW, MEDIUM, HIGH, with probability of defect-proneness thresholds (e.g. LOW with probability being less than 50%, MEDIUM with probability between 50% and 75%, HIGH with probability being greater than 75%). Dione uses these custom levels and predicts files' defect-proneness based on these levels. This approach helps to prioritize certain software modules that are most probably defect-prone among others.

In Figure 5, a screenshot from prediction results is shown. While displaying predictions, a hierarchical tree-like structure showing all software modules in the test set from package level to method level are presented. For each software module, probability of defect-proneness and user-defined prioritization levels are reported. Furthermore in top right region of Figure 5, a multivariate analysis of software metrics and defects in historical data is also listed.

We will predict the post release defects of Eclipse for version 3.2 during the demo. We will show the defect prediction configuration step initially. In this step we will define the training data, machine learning algorithms that will be used by the model and the schedule parameters of the defect prediction run. Afterwards, we will schedule the defect prediction batch job and show the task monitoring features within Dione.

Finally, we will show the defect prediction output and compare the findings of Dione with the actual post-release defect data of Eclipse.

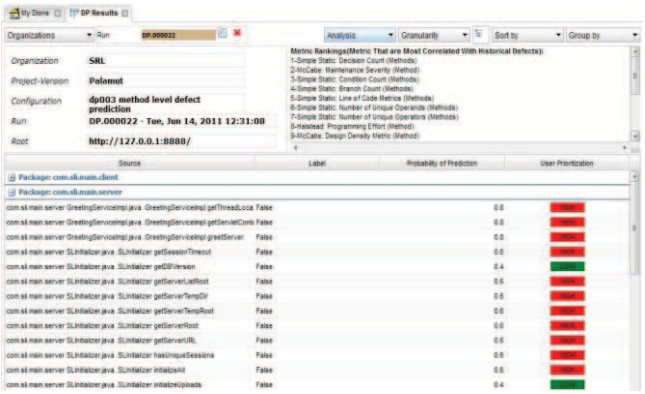

Figure 5 Defect Prediction Output of Dione