## Letter of Transmittal

Contract Management Branch for Center Operations and Space
Attn: Grants Officer, MS:241-1
Moffett Field, CA 94035-1000

1. **Legal name and address of organization**

   University of Hawaii
   Office of Research Services
   2530 Dole Street, Sakamaki D-200
   Honolulu, HI 96822

   Taxpayer ID: 99-6000354
   Cage No: 0W411
   DUNS No: 965088057

2. **Type of organization:**
   Educational

3. **Principal Investigator:**
   Philip M. Johnson, Professor
   Department of Information and Computer Sciences
   University of Hawaii
   1680 East-West Rd, POST 317
   Honolulu, HI 96822
   808 956-3489

4. **Title of proposal:**
   Supporting development of highly dependable software through incremental, automated, in-process, and individualized software measurement validation.

5. **Other government agency to which this proposal has been submitted:**
   This proposal is submitted as part of the joint NSF/NASA Highly Dependable Computing and Communication Systems Research program.

6. **Name of NASA individual with whom preliminary discussion has been held:**
   Dr. Michael Lowry
   NASA Ames Research Center, M/S 269-2
   Moffett Field, CA 94035
   650-604-3369

7. **Cooperative agreement number (if for renewal or continuation):**
   None. This is a new submission.

8. **Date of submission:** 11/18/2003
   **Desired starting date:** 12/01/2003
   **Duration of project:** 12/01/2003 - 8/01/2006

9. **Name and signature of responsible person authorized to represent and contractually obligate the offerer:**

   _____
   Kevin Hanaoka, Interim Director, ORS

# Contents

# 1 Abstract

Highly dependable software is, by nature, predictable. For example, one can predict with confidence the circumstances under which the software will work and the circumstances under which it will fail. Empirically-based approaches to creating predictable software are based on two assumptions: (1) historical data can be used to develop and calibrate models that generate empirical predictions, and (2) there exists relationships between *internal* attributes of the software (i.e. immediately measurable process and product attributes such as size, effort, defects, complexity, and so forth) and *external* attributes of the software (i.e. abstract and/or non-immediately measurable attributes, such as "quality", the time and circumstances of a specific component's failure in the field, and so forth). *Software measurement validation* is the process of determining a predictive relationship between available internal attributes and correspondingly useful external attributes and the conditions under which this relationship holds.

The general objective of this research is to design, implement, and validate software measures within a development infrastructure that supports the development of highly dependable software systems. The measures and infrastructure are designed to support dependable software development in two ways: (1) They will support identification of modules at risk for being fault-prone, enabling more efficient and effective allocation of quality assurance resources, and (2) They will support incremental software development through continuous monitoring, notifications, and analyses. Empirical assessment of these methods and measures during use on the Mission Data System project and related testbeds at Jet Propulsion Laboratory will advance the theory and practice of dependable computing and software measurement validation and provide new insight into the technological and methodological problems associated with the current state of the art.

# 2 Project Description

## 2.1 Motivation

Highly dependable software is, by nature, predictable. With respect to behavior, one should know with confidence the situations under which the software will behave correctly and the situations under which it will fail. With respect to development, one should know with confidence how to assess the dependability of existing components and how to improve the dependability of components that are not sufficiently dependable. Finally, with respect to planning, one should know with confidence the cost and resources required to obtain the desired level of dependability. Unpredictable systems create dependability risks with respect to behavior, development, and planning, leading to dependability failures involving availability, reliability, safety, confidentiality, integrity, and maintainability [2].

Empirically-based approaches to creating predictable software are based on two assumptions: (1) historical data can be used to develop and calibrate models that generate empirical predictions, and (2) there exists relationships between *internal* attributes of the software (i.e. immediately measurable process and product attributes such as size, effort, defects, complexity, and so forth) and *external* attributes of the software (i.e. abstract and/or non-immediately measurable attributes, such as "quality", the risk of a specific component's failure in the field, and so forth). *Software measurement validation* is the process of determining a predictive relationship between available internal attributes and correspondingly useful external attributes and the conditions under which this relationship holds [13]. A validated software measure thus predicts things of interest about the future of a software development project and its resulting products based upon information at hand. Measurement validation is particularly important for quality: the ISO/IEC international standard on software product quality states that "Internal metrics are of little value unless there is evidence that they are related to external quality" [30].

Many recent attempts to create valid software measures focus on prediction of a specific external measure: the fault-proneness of individual modules after release. This focus is due to findings that the distribution of faults across modules tends to be non-uniform [29, 14, 28]. For example, 47 percent of the faults found by users of OS/370 were associated with only four percent of the modules. A valid measure for fault-proneness identifies a subset of modules at higher risk for post-release faults, and can be used to achieve more efficient and effective deployment of quality assurance resources. Predictable module-level fault-proneness is one of many possible ways to reduce dependability risks through valid software measures.

Recent research indicates the feasibility of creating valid software measures in both academic settings [3, 12, 31, 9, 11] and industrial settings [8, 5, 25, 17]. Despite these encouraging findings, broad application of valid measures is hampered by a number of problems including lack of empirical evaluation in production settings, the possibility of confounding variables that reduce the predictive value or generality of the measure, the unknown impact of defect contagion and inheritance structures, and the overall cost of measure validation [4, 13, 6, 14].

This research project includes technological and methodological innovations designed to address these issues and advance the state of the art in both highly dependable computing and software measurement validation. It will do so through a set of measurement validation projects involving the Mission Data System at Jet Propulsion Laboratory and the SCRover testbed adaptation of that software architecture.

## 2.2 Objectives and Scientific Hypothesis

The general objective of this research is to design, implement, and validate software measures within a development infrastructure that supports the development of highly dependable software systems. The measures and infrastructure are designed to support dependable software development in two ways: (1) They will support identification of modules at risk for being fault-prone, enabling more efficient and effective allocation

of quality assurance resources, and (2) They will support incremental software development through continuous monitoring, notifications, and analyses. Empirical assessment of these methods and measures will advance the theory and practice of dependable computing and software measurement validation and provide new insight into the technological and methodological problems associated with the current state of the art.

We will pursue this general objective through the following seven specific objectives:

(1) We will enhance the Hackystat system [18] with mechanisms for unobtrusive, continuous, and automatic collection of five classes of internal product and process measures of the Mission Data System (MDS) architecture for reusable flight control software developed by Jet Propulsion Laboratory (JPL);

(2) We will build risk models for predicting fault-proneness in MDS modules based upon current best practices [3, 8, 12, 7];

(3) We will use the in-process alert mechanisms of Hackystat to integrate these risk models into ongoing development, so that developers obtain "just-in-time" notification of modules at risk for pre-release or post-release faults.

(4) We will replicate objectives (1)-(3) on the SCRover testbed software, and compare the risk models obtained for the MDS architecture to the testbed;

(5) We will perform quantitative evaluations of the accuracy of the models and qualitative evaluations of the usefulness of the alert-based integration for ongoing development;

(6) We will make the results broadly available by publishing the open source Hackystat software (comprising sensors, risk models, and alerts) and by publishing the model definitions and our evaluation results in peer-reviewed journal articles. (The actual process and product data will remain proprietary to JPL). Our successes and failures will facilitate future advances in dependable computing and software measurement validation.

(7) We will enhance the infrastructure for education by developing software engineering curriculum modules that enable academic and industrial practitioners to learn how to use Hackystat to support software measurement validation for highly dependable computing in a classroom or industrial setting. The open source Hackystat software will enhance the infrastructure for research by facilitating replication and enhancement of this research.

These objectives combine to support the empirical testing of the following scientific hypothesis: *Validated risk models for predicting fault-proneness can improve the dependability of MDS and SCRover testbed software.*

### 2.3 Hackystat: a system for software measurement validation

Hackystat is a client-server, sensor-based web service that provides a generic framework for unobtrusive collection and analysis of process and product measurements. The system was initially released in July, 2001, and is currently in active use at three sites. The design of Hackystat is influenced by our experiences developing instrumented applications for computer-supported cooperative work and software quality assurance over the past ten years [19, 20, 32, 22, 23]. We have learned that in-process, fine-grained measures have great potential for providing useful insight into work processes in general and measurement validation in particular, but that adoption is problematic unless the overhead of data collection and analysis is extremely low. This appears to be true regardless of the potential (long-term) benefits of the data.

Hackystat is designed to be used by developers in two contexts, which we call *measure-generation* mode and *measure-driven* mode. Among other things, measure-generation mode supports the discovery of valid software measures, while measure-driven mode makes the results of those measures available to developers in a "just-in-time" fashion.

To begin, assume that a valid measure is already available and the system is being used in measure-driven mode. (In reality, the system can be used in both modes simultaneously, with some valid measures used for analysis and prediction while others are being collected for subsequent exploration.) To start, developers

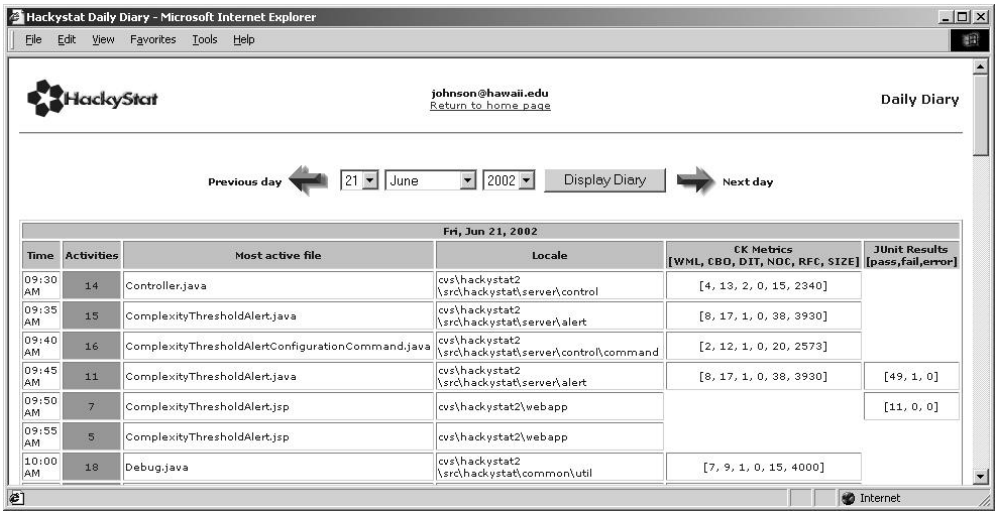


Figure 1: *Individual developer process and product metrics at five minute intervals. The activity sensor monitors developer actions, and infers the file that was the "most active" object of effort during each five minute interval. This file's directory structure provides the "locale", or region in the system where effort was occurring. When the most active file is a Java file, a second sensor calculates its Chidamber-Kemerer metrics from the .class file associated with the Java file, as well as its size in bytecodes. A third sensor monitors invocation of the JUnit regression testing tool and provides a summary of the results to this page.*

download and attach custom built sensors to their development tools, such as their editors, unit testing tools, configuration management system, build tool, and so forth. These sensors unobtrusively monitor individual developer activities and send data on both process and product to a web server. Once a day, a variety of analysis mechanisms are invoked over the accumulated data. If these analyses discover "anomalies", the web service sends an alert via an email to the developer informing them that an anomaly has been discovered. The email also includes an URL to a page on the server that provides detailed empirical data about the nature of the anomaly. The developer can use this link to learn more about the problem, and also to provide feedback to the system on whether the alert was valid, i.e. that anomaly actually provides useful information to the developer.

The current implementation of Hackystat includes three kinds of sensors: one collects "activity" data on developer effort and focus of attention; another collects the current values of the Chidamber-Kemerer object oriented complexity metrics associated with the Java class file being worked on by the developer; and the third sensor collects pre-release defect data (through the results of running JUnit regression tests.) The "Daily Diary" page provides a summary of the data collected by each of these sensors for a specific developer during a single day, broken down into five minute intervals.

Figure 1 illustrates a portion of one such Daily Diary. For each five minute interval in which sensor data is recorded, the page displays the "most active file" that the developer interacted with, as well as its "locale" or region of the system (normally the directory structure with volume information discarded.) When the most active file is a Java file, the page displays five of the associated Chidamber-Kemerer metrics (Weighted Methods per Class, Coupling between Objects, Depth of Inheritance Tree, Number of Children, and Response for Class) as well as the size of the class [1]. Finally, the Daily Diary shows when pre-release unit tests are run and their results.

[1] The size of a Java class is measured in bytecodes, which preliminary studies indicate to be very highly correlated ($r^2 = .93$) with non-comment source lines of code. On average, 40 bytecodes are required for each non-comment source line of code in a Java class.

This Daily Diary page from the current implementation illustrates the ability of Hackystat to record and analyze both process measures (what file the developer is working on and when she is working on it; when are unit tests invoked) and product measures (what is the complexity of the class under development, how is it changing, and what is the quality of the current state of the system with respect to its performance during unit testing.) There are many ways to exploit these measures; for this example of measure-driven mode we will illustrate the use of the "Complexity Threshold Alert".

The complexity threshold alert monitors the values of the six complexity measures tracked for each Java class and sends the developer an email when threshold values for some combination of the measures exceeds predetermined values. To obtain details on this anomaly, the developer retrieves the URL in the email, which provides details on the complexity values. If the alert algorithm was validated for fault-proneness, then these classes are candidates for additional quality assurance, such as Inspection, redesign/refactoring, or additional testing.

This example of automated collection, analysis, and feedback of complexity data, while simple, nevertheless supports a central claim of this research proposal: that Hackystat provides efficient and effective infrastructure for the development of valid software measures. In this case, the hypothesized valid measure consists of a useful, predictive relationship between a set of internal complexity measures and the external measure of post-release fault-proneness. The threshold value settings could be determined by adapting published valid measures [3, 8], through developer intuition, or through percentile rankings that identify the most complex classes relative to the system as a whole. Note that the current Hackystat sensors can support the development of a more sophisticated measure that augments the internal product measures of complexity with additional internal process measures including the effort expended on the class by the developer and the pre-release defect history.

Hackystat in measure-driven mode differs in four ways from prior approaches to software measurement validation:

First, validation activities are *automatic*. Once the sensors are installed and the analysis mechanism activated, the developer does not have to manually collect or periodically "poll" the data analysis functions; the system will instead "interrupt" the developer when an event of interest occurs.

Second, validation measures can be *individualized*. Each developer's data can be used to generate a personalized measure and/or alert scheme that reflects their activities and development style.

Third, measure application is *continuous*. In the above example, developer activities are monitored and product and process measures are reanalyzed each day as the developer changes old classes or implements new ones.

Fourth, the validation is *in-process*. Instead of gathering data from a set of developers on one project and hoping the measure applies to later projects with potentially different developers, Hackystat facilitates development of valid measures that feed back immediately into the current development process.

In contrast to measure-driven mode, in which one or more measures are assumed to be valid and thus useful for providing insight to developers on the status or trajectory of development, measure-generation mode is a more exploratory process in which the values of internal measures maintained by Hackystat are related to external measures (which may or may not be maintained by Hackystat.) The central objective of this research, which is to design, implement, and validate software measures that support the development of highly dependable software systems, will be pursued by defining and pursuing a methodology for measure-generation mode at Mission Data Systems and its subsequent application in measure-driven mode. This methodology will be described in Section 2.5; the next section introduces the Mission Data System development organization and development procedures necessary to motivate the methodology.

## 2.4 The Mission Data System at Jet Propulsion Laboratory

Until recently, deep space missions tended to be one-of-a-kind, with distinct science objectives, instruments, and mission plans. For example, when Jet Propulsion Laboratory launched six missions in six months between October 1998 and March 1999, there was no common framework for developing mission software and little software reuse. The goal of the Mission Data System project is to develop a set of software architectures that accommodate the complexities of future mission requirements, including architectures for flight, ground, and test data systems. MDS facilitates software reuse, expands autonomous capabilities, and enables infusion of new software technologies. Users of MDS will benefit from a unified architectural framework for building end-to-end flight and ground software systems, along with executable example uses of those frameworks running a simulated mission [1]. MDS currently consists of approximately 5,700 C++ classes and 548,000 non-comment lines of code.

The MDS software has very high dependability requirements, and represents significant advances in the areas of software design, construction, and operation. Some of the architectural characteristics creating dependability challenges include: (1) the need for correct operation over a very large behavior space; (2) mission commands expressed as high level "goals" that are expanded into control sequences by on-board software; (3) architectural components that include reasoning components interacting with internal models of spacecraft structure and behavior.

The MDS development group employs a highly automated incremental development process with extensive tool support for configuration management, system build, and automated testing [27]. In general, each high-level development increment is organized into a set of work plans according to their impact on six areas: simulation, flight, transport, ground, framework, and test. Work plans are broken down into Implementation Tasks, which are further broken down into Change Packages. Change packages specify the actual files requiring modification to accomplish the enhancement and the responsible developer(s), and are the units of work input to the AllFusion Harvest Change Manager system [15]. Harvest maintains a workflow state model that tracks the progress of the Change Package as it goes through the states of "Dev Waiting", "Dev", "Dev Complete", "Build Queue", "Build Test", "Integration Test", "Test Complete", and "Release".

Dependability-related problems in the MDS development process can be manifested in many ways, but three events are particularly important to this research project. First, a Change Package may fail during Build Test or Integration test. This normally indicates a pre-release fault, and is one of the situations leading the second event: the creation of an Internal Anomaly Report (IAR). Both build failures and IARs represent pre-release faults in the software. Once MDS is delivered to a customer and is under adaptation, use during either mission software development or field use may result in the discovery of additional faults. Each of these post-release faults will result in the third event: the creation of an Anomaly Report (AR).

Figure 2 provides an abstract view of the relationship between the MDS architecture development effort, the Mission software development projects that will utilize MDS as a component, and the ultimate field use of the mission software. The goal of the diagram is to point out how pre-release faults (represented by Internal Anomaly Reports) can be generated during MDS development or Mission software development, and that MDS post-release faults (represented by Anomaly Reports) can occur during either Mission software development or resulting field use. (Although the Mission software development process is not required to use MDS build technology and workflow representations such as IARs and ARs, they are employed here for simplicity's sake.)

The MDS project is an ideal testbed for the proposed research on highly dependable software. First, both the requirements for dependability and the costs of failure are extremely high, which means the organization is motivated and open to innovative approaches. Second, the current level of development process automation means that useful internal measures are readily available for collection by Hackystat sensors. Finally, the presence of two concurrent cycles of development (MDS and the Mission software adaptation) creates the possibility of combining internal measures from multiple project sources to generate new kinds of quality

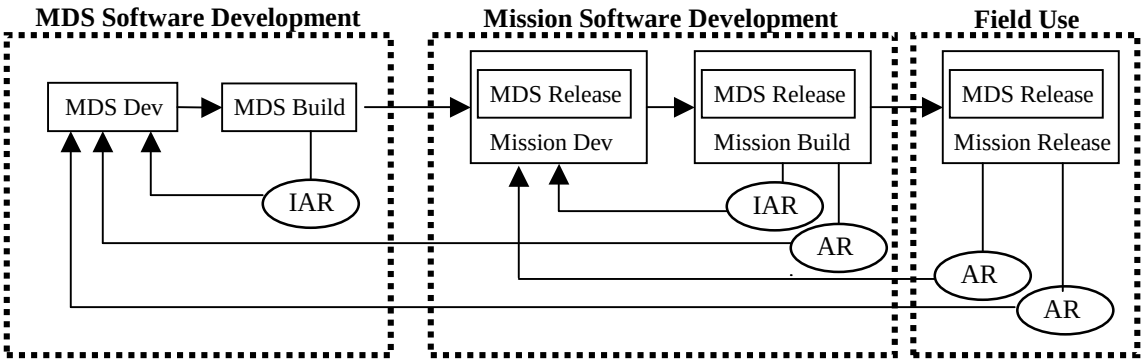


Figure 2: *MDS software development yields a release of MDS, which becomes a component during development of a specific mission's software. Internal Anomaly Reports (IARs) represent pre-release faults within a development process, while Anomaly Reports (ARs) represent post-release faults.*

assurance models, as will be discussed next.

## 2.5 Research plan

### 2.5.1 Overview of valid software measure development for MDS

In the context of the MDS development process, "perfection" means satisfying customer requirements while simultaneously preventing all fault creation, as measured by the cessation of all Internal Anomaly Reports and Anomaly Reports. Fortunately, a perfect development process is not essential for very high dependability. Instead, high dependability can be pursued by improving the predictability of fault-proneness, as measured by improving the ability of the development organization (either MDS or a Mission-specific one) to predict characteristics of the future occurrence and nature of Internal Anomaly Reports and Anomaly Reports based upon the past history and current state of development.

Figure 3 illustrates the proposed approach to improving predictability and thus dependability in the current MDS development context. It builds off the representation of faults as IARs and ARs as illustrated in Figure 2, and uses these in concert with internal measures for five software measurement validation experiments.

The top third of Figure 3 is similar to that in Figure 2, except the generic "Mission" software development in Figure 2 is replaced by an actual HDCP Testbed software development project [26]. The SCRover testbed project is intended to simulate most aspects of an actual Mission development project, thus serving as a basis for dependability related research and to demonstrate the feasibility of MDS for actual mission development. Testbed development will result in a functioning software system that can be deployed and evaluated in field conditions.

While the diagram illustrates a linear process for simplicity's sake, in reality the MDS and Testbed development processes are concurrent and interdependent: Testbed development may yield MDS ARs that lead to new releases of MDS that are incorporated into ongoing Testbed development. One research question to be explored is the impact of new releases on the predictive models and determining the circumstances under which recalibration becomes necessary.

### 2.5.2 Five software measurement validation experiments

The circled "1" in Figure 3 represents the first software measurement validation experiment. The left-most parenthesized list indicates a set of internal measures of MDS development (size, complexity, coverage, effort, and time in phase), while the adjacent parenthesized list indicates a set of external measures available

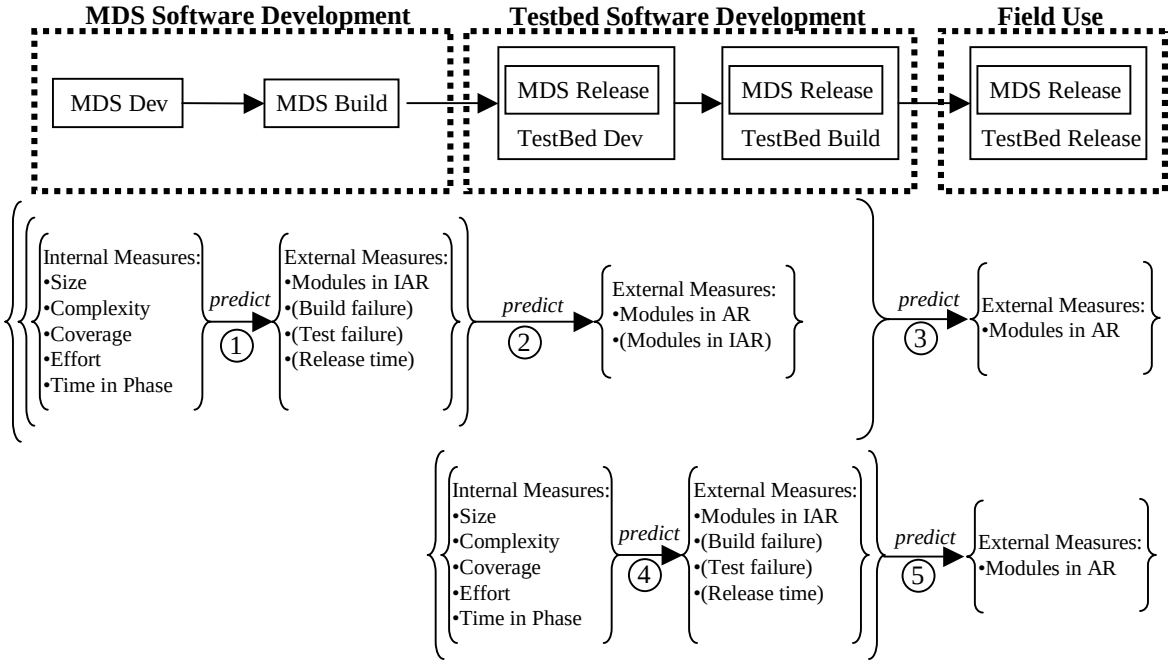


Figure 3: *The five measure validation experiments for the MDS and SCRover testbed development projects. The top of the diagram shows that MDS development yields a release that is used as a component by the testbed development process, which yields a release of a testbed system that is deployed in the field. The middle of the diagram illustrates three measurement validation experiments using dependability attributes based upon MDS process and product data to predict MDS, testbed, and Field defects. The bottom of the diagram illustrates a replication within the testbed development environment. (Prediction of external measures in parentheses will be investigated but are less important for dependability.)*

later in development (the most important of which is the modules or files implicated in an Internal Anomaly Report, though build failure, test failure, and release time can also be analyzed). Using the methodology described in Section 2.5.4, we will attempt to develop a valid software measure that relates one or more of the MDS internal measures to the external measure of pre-release fault-proneness as evidenced by IARs. We will also explore the use of MDS internal measures to predict other external measures such as Build failure, Test failure, and the calendar time required for a Change Package to be released, although these are somewhat less important for dependability.

Recall from Figure 2 that while the MDS project can generate pre-release fault information, post-release fault information comes from a Mission-specific development process as well as from actual field use. The second software measurement validation experiment uses MDS Anomaly Report data from the Testbed development process. We will attempt to develop a valid software measure that uses both internal and external measures from within MDS development to predict the occurrence of MDS Anomaly Reports during Testbed development. Such a measure will help MDS developers identify in advance which modules are more likely to fail during adaptation to a specific mission context. This experiment is illustrated with the circled “2” in Figure 3 through the parenthesized lists of internal and external measures for MDS that predict two external measures of Testbed software development. (The parenthesized external measure, Modules in IAR, indicates that the analysis may yield a measure that helps identify fault-prone Testbed modules, although this is less important than identifying fault-prone MDS modules.)

The circled “3” in Figure 3 represents the third software measurement validation experiment. It combines data from MDS and Testbed development to predict those modules responsible for post-release defects during

field use of the Testbed system.

The fourth and fifth software measurement validation experiments are replications of the first and second projects described above. In the fifth experiment, we will collect measures from Testbed software development and attempt to define a valid software measure that uses internal Testbed measures to predict pre-release module fault-proneness, as evidenced by IAR reports (or whatever representation is used in the Testbed development process). In the sixth experiment, we will use both internal and external Testbed measures to attempt to define a valid software measure for predicting module fault-proneness in the field. These last two experiments are represented by the circled "4" and "5" in Figure 3.

### 2.5.3 Dependability attributes

The set of internal and external measures discussed above can also be viewed as the set of "dependability attributes" to be studied in this research, consisting of:

- *Size*, measured as non-comment source lines of code;
- *Complexity*, measured using one or more of the standard Chidamber-Kemerer object oriented metrics (CBO, WMC, RFC, or LCOM);
- *Coverage*, measured as the percentage of lines exercised by the test cases (statement coverage);
- *Effort*, measured as developer-hours;
- *Time in phase*, measured by calendar days spent by a work package in a specified phase;
- *Modules in IAR*, measured as the number of distinct modules referenced by Internal Anomaly Reports;
- *Modules in AR*, measured as the number of distinct modules referenced by Anomaly Reports;
- *Build Failure*, measured as the percentage of build attempts that did not succeed;
- *Test Failure*, measured as the percentage of test case invocations that did not succeed;

### 2.5.4 Experimental methodology

Each of the five validation experiments consists of four phases: Sensor Building, Baseline Data Collection, Model Construction, and Model Evaluation.

During the first phase, *Sensor Building*, we will implement any Hackystat sensors not yet available for collecting the measures of interest to the experiment. Sensors are specific to the tools and languages used by the developer group. For MDS, we will augment our current set of sensors for JUnit, Java, Emacs, and JBuilder with new sensors for C++ and the Harvest configuration management tool. The additional sensors required for the Testbed project will depend upon the development group selected for that project. Sensor building effort will be front-loaded: we expect to spend a significant amount of time during the first two years on Sensor Building, but relatively little time during the second two years.

During the second phase, *Baseline Data Collection*, we will use the deployed Hackystat sensors to begin automated collection of both the internal and external measures for one external release of the target system. Internal measures include the following product and process measures of a single module: the Chidamber-Kemerer metrics WMC, DIT, NOC, RFC, and CBO [10]; Briand's method invocation import coupling [8]; size (non-comment source lines of code); unit test coverage; developer effort; unit test failure rate; build failure rate; and interval time in build. External measures of a single module include: the number and severity of IARs related to this module; the number and severity of ARs related to this module. A module will be either a C++ class (in the case of MDS) or a Real-Time Java class (in the case of the Testbed project).

From an experimental point of view, the internal measures correspond to independent variables, and the external measures correspond to dependent variables.

During the third phase, *Model Construction*, we will analyze the baseline data collected for one or more external releases of the system and construct risk models that predict fault-proneness (i.e. at least one IAR or AR of a given severity level) for a module given the values of the internal measures (i.e. the product and process measures listed above). Our approach will follow best practice guidelines for validating software product metrics [12]. We plan to construct risk models based upon logistic regression which compute the expectation of a component being fault-prone given the values of selected internal measures (independent variables). Logistic regression is a standard classification technique based upon maximum likelihood estimation [16, 24]. To select the internal measures to be used in the risk models, we will use Baseline data to assess the strength of the relationship of each internal measure to the external measure after controlling for size. For models based upon logistic regression, the strength of the relationship is measured by the change in the odds ratio. To increase the stability of the model, we will seek a set of internal measures that are strongly correlated to the external measure while weakly correlated to each other, thus reducing collinearity. We will investigate the confounding variable of developer experience by building individualized models based on each developer's process and product data and comparing that to a model based upon an aggregate of all product and process data.

During the fourth phase, *Model Evaluation*, each model will be subjected to a goodness of fit analysis on the Baseline data. This analysis checks to see that the model has reasonable predictive accuracy for the historical data used to generate it. Following this retrospective evaluation, the model's usefulness will be evaluated in the context of the next external release cycle of development by being incorporated into Hackystat as one or more alerts. Each alert will augment the constructed model with a threshold probability value. When this value is exceeded, Hackystat will generate an email to the developer that the corresponding module has been predicted to be fault-prone. The corresponding URL will show the values of the internal variables that gave rise to the alert. Evaluation of the model in the context of ongoing development will have both quantitative and qualitative components. The quantitative component will compare the model's predictions of fault-proneness to the actual ARs and IARs subsequently generated and test for significance using the PRESS (Predictive Sum of Squares) statistic [33]. The qualitative component will use buttons on each page displayed by alert URLs that allow the developer to give feedback to the system on the utility of that particular alert. In combination with developer interviews, qualitative feedback will provide insight into the perceived utility of the predictions to the developers, and ways in which in-process fault-proneness prediction can be improved.

### 2.5.5 Deliverables

The experiments will result in three classes of deliverables: technology, methodology, and measurement validation hypothesis testing. The principal technology deliverable is the enhanced version of the open source Hackystat system. The proposed research will expand Hackystat with a significant number of additional sensors for new tools and languages, as well as alerts built upon logistic regression modeling. As the research proceeds, we may investigate sensors for additional internal measures such as module-level documentation quality, which may predict certain types of faults for systems such as MDS which are intended to be used as components in other systems. The research will yield experience using a continuous, automated, in-process, and individualized methodology for measurement validation which has been used in five different experimental settings. This methodology will be made available for technology transfer within NASA as well as incorporated into software engineering educational settings using Hackystat. The final deliverable will be the results of hypothesis testing for each experiment, yielding insight into the kinds of internal measures useful for predicting fault-proneness (if any), the effect of developer experience (if any), and the development contexts in which the proposed approach to measurement validation is effective and usable for

reducing dependability risks.

### 2.5.6 Schedule and Milestones

This research project consists of five experiments, each of which has four phases. We have already begun work on the first two phases (Sensor Building and Baseline Data Collection) for the first experiment, and have successfully built both sensors and analysis mechanisms for the Harvest system. A report on the results of this project is currently available [21]. Experiments subsequent to the first one can reuse and/or adapt sensors and models created during the first experiment, and this should reduce the interval time for experiments 2-5 to twelve to eighteen months. Experiments will be run concurrently after an initial startup period. Thus, we expect to complete the first two experiments in Fall, 2005, with the subsequent experiments completed every six months thereafter (Experiment 3, Spring 2005; Experiment 4, Fall 2005; Experiment 5, Spring 2006).

# 3 References


[1] MDS: The strategic advantage for future flight projects. Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, January 2001.

[2] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of dependability. Technical Report UCLA CSD Report 010028, University of California, Los Angeles, 2001.

[3] Victor Basili, Lionel Briand, and Walcelio Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10), 1996.

[4] Saida Benlarbi, Khaled El Emam, and Nishith Goel. Issues in validating object-oriented metrics for early risk prediction. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, November 1999.

[5] Aaron Binkley and Stephen Schach. Metrics for predicting run-time failures and maintenance effort: Four case studies. *CrossTalk*, August 1998.

[6] S. Biyani and P. Santhanam. Exploring defect data from development and customer usage of software modules over multiple releases. In *Proceedings of the International Symposium on Software Reliability Engineering*, 1998.

[7] Barry Boehm. *Software Risk Management*. IEEE Computer Society Press, 1989.

[8] Lionel Briand, Jurgen Wust, and Hakim Lounis. Replicated case studies for investigating quality factors in object-oriented designs. *Journal of Empirical Software Engineering*, 2001.

[9] M. Cartwright. An empirical view of inheritance. *Information and Software Technology*, 40, 1998.

[10] Shyam Chidamber and Chris Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6), June 1994.

[11] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood. Evaluating inheritance depth on the maintainability of object oriented software. *Journal of Empirical Software Engineering*, 1(2), 1996.

[12] Khaled El Emam. A methodology for validating software product metrics. Technical Report NRC/ERB-1076, National Research Council of Canada, June 2000.

[13] Khaled El Emam. Object-oriented metrics: A review of theory and practice. Technical Report NRC/ERB-1085, National Research Council of Canada, June 2000.

[14] Norman Fenton and Niclas Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8), August 2000.

[15] Allfusion harvest change manager. Computer Associates, Inc. (http://www.ca.com/).

[16] D. Hosmer and S. Lemeshow. *Applied Logistic Regression*. John Wiley and Sons, 1989.

[17] John Hudepohl, Stephen Aud, Tahgi Khoshgoftaar, Edward Allen, and Jean Mayrand. Emerald: Software metrics and models on the desktop. *IEEE Software*, 13(5), September 1996.

[18] Philip M. Johnson. Hackystat system. http://csdl.ics.hawaii.edu/Research/Hackystat/.

[19] Philip M. Johnson. Experiences with EGRET: An exploratory group work environment. *Collaborative Computing*, 1(1), January 1994.

[20] Philip M. Johnson. Design for instrumentation: High quality measurement of formal technical review. *Software Quality Journal*, 5(3):33–51, March 1996.

[21] Philip M. Johnson. The Hackystat JPLBuild system: Overview and initial results. Technical Report CSDL-03-07, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, October 2003.

[22] Philip M. Johnson and Anne M. Disney. A critical analysis of PSP data quality: Results from a case study. *Journal of Empirical Software Engineering*, December 1999.

[23] Philip M. Johnson, Carleton A. Moore, Joseph A. Dane, and Robert S. Brewer. Empirically guided software effort guesstimation. *IEEE Software*, 17(6), December 2000.

[24] Taghi Khoshgoftaar and Edward Allen. Logistic regression modeling of software quality. *International Journal of Reliability, Quality and Safety Engineering*, 6(4), 1999.

[25] Jean Mayrand and Francois Coallier. System acquisition based on software product assessment. In *Proceedings of the 18th International Conference on Software Engineering*, 1996.

[26] Kenny Meyer and Daniel Dvorak. Mission data system HDCP testbed: An end-to-end platform for improving the dependability of state-based mission software built from component frameworks. Technical report, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 2002.

[27] Kenny Meyer and Carl Puckett. Introduction to the MDS process. Technical report, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 2001.

[28] K-H Moller and D. Paulish. An empirical investigation of software fault distribution. In *Proceedings of the First International Software Metrics Symposium*, 1993.

[29] G.J. Myers. *The Art of Software Testing*. John Wiley and Sons, New York, 1979.

[30] International Standards Organization. ISO/IEC 14598-2 information technology – product evaluation, part 1, general overview.

[31] Ganesh Pai and Joanne Dugan. Enhancing software reliablity estimation using bayesian belief networks and fault trees. In *Proceedings of the 12th International Symposium on Software Reliability Engineering*, November 2001.

[32] Adam A. Porter and Philip M. Johnson. Assessing software review meetings: Results of a comparative analysis of two experimental studies. *IEEE Transactions on Software Engineering*, 23(3):129–145, March 1997.

[33] R. Szabo and T. Khoshgoftaar. Modeling software quality in an object-oriented software system. In *Proceedings of the Annual Oregon Workshop on Software Metrics*, 1995.

# 4 Management Approach

This project will involve collaborations between three groups:

1. *University of Hawaii*. This group consists of the Principal Investigator (Philip Johnson), the co-Principal Investigator (Dan Port, Assistant Professor of Information Technology Management at the University of Hawaii), and the four graduate student researchers to be funded under this cooperative agreement;

2. *Mission Data System*. This group consists of management and developers in the Mission Data System group at Jet Propulsion Lab. Over the past year, we have interacted most often with Kenny Meyer.

3. *SCRover*. This group consists of management and developers of this testbed system. Over the past year, we have interacted most often with Barry Boehm and Alex Lam.

We expect that our collaboration will involve approximately 3-4 meetings per year at JPL or USC, augmented by more frequent teleconferences and emails as circumstances require.

# 5 Personnel

## 5.1 Philip Johnson

#### Degrees

Ph.D. in Computer Science, University of Massachusetts. 1990
M.S. in Computer Science, University of Massachusetts. 1985
B.S. in Biology, B.S. in Computer Science, University of Michigan. 1980

#### Academic Appointments

*Professor*, Information and Computer Sciences, University of Hawaii. 2001—present
*Associate Professor*, Information and Computer Sciences, University of Hawaii. 1995—2001
*Senior Research Fellow*, Distributed Systems Technology Centre, University of Queensland, Australia, 1997.
*Assistant Professor*, Information and Computer Sciences, University of Hawaii. 1990—1995

#### Selected Publications

P. M. Johnson and C. A. Moore and J. A. Dane and R. S. Brewer, *Empirically Guided Software Effort Guesstimation*. IEEE Software, Vol. 17, No. 6, December 2000.

P. M. Johnson and A. M. Disney, *A Critical Analysis of PSP Data Quality: Results from a Case Study*. Journal of Empirical Software Engineering, Volume 4, December, 1999.

P. M. Johnson, *Reengineering Inspection*. In Communications of the ACM, Volume 41, No. 2, February, 1998.

P. M. Johnson and D. Tjahjono, *Does Every Inspection Really Need A Meeting?*, In Journal of Empirical Software Engineering, Volume 4, No. 1, January 1998.

A. A. Porter and P. M. Johnson, *Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies*. In IEEE Transactions on Software Engineering, vol. 23, no. 3, March 1997.

P. M. Johnson and A. M. Disney, *The Personal Software Process: A Cautionary Case Study*. In IEEE Software, Volume 15, No. 6, November, 1998.

P. M. Johnson, *Design for Instrumentation: High Quality Measurement of Formal Technical Review*. Software Quality Journal, Volume 5, March, 1996.

#### Professional Activities

*Editorial Board*, IEEE Transactions on Software Engineering, 2000-present.
*Program Chair*, International Software Engineering Research Network Annual Meeting, 2000.
*Advisory Board Member*, International Journal of Computer Supported Cooperative Work
*Program Committees*:

XP/Agile Universe, 2004.
International Software Metrics Symposium, 2002, 2003, 2004.
International Symposium on Empirical Software Engineering, 2002.
European Conference on Computer Supported Cooperative Work, 1997, 1999.
Software Architectures for Cooperative Systems Workshop, 1994
CSCW Tools and Technologies Workshop, 1992, 1993

## 5.2 Other personnel

In addition to the Principal Investigator, other personnel associated with this project include:

- *Dan Port*, Assistant Professor of Information Technology Management, University of Hawaii College of Business Administration.
- *4 Graduate Students, Information and Computer Sciences.* Currently we have two Ph.D. students and two M.S. students working on this project.

# 6 Facilities and Equipment

The facilities used for this research will be based primarily at the University of Hawaii, in the Collaborative Software Development Laboratory of the Department of Information and Computer Sciences. Additional facilities will be used for occasional meetings based at the Mission Data System at Jet Propulsion Lab and at the University of Southern California for SCRover collaborations.

Equipment will include PCs, networking equipment, and other software development infrastructure at the University of Hawaii.

# 7 Proposed Costs

## 7.1 Cost categories and amounts

| Cost Category | 2004 | 2005 | 2006 | Category Total |
|---|---|---|---|---|
| Salaries (Senior Personnel) | $9,609 | $9,609 | $9,609 | $28,827 |
| Salaries (Graduate Assistants) | $33,648 | $33,648 | $33,648 | $100,944 |
| Fringe Benefits | $8,748 | $8,748 | $8,748 | $26,244 |
| Travel | $4,000 | $4,000 | $4,000 | $12,000 |
| Materials and Supplies | $2,500 | $2,500 | $2,500 | $7,500 |
| Indirect Costs | $21,237 | $21,237 | $21,237 | $63,711 |
| **Yearly Total** | $79,741 | $79,741 | $79,741 | $239,223 |

## 7.2 Cost category descriptions

The following cost category descriptions are based upon co-funding for the three years of this project from the National Science Foundation under the Highly Dependable Computing and Communication Systems program. Thus, the amount requested in this proposal for Graduate Assistants is actually only half of the amount required to fully fund these four positions during the project.

### Salaries

The project budget provides salary support for the principal investigator (two summer months) and four graduate research assistants (11 months) for each of the four years. The principal investigator will perform or supervise all major system design enhancements and empirical studies, manage and train the graduate student assistants, and supervise activities at JPL.

### Travel

The project budget provides funds for four trips per year from Hawaii to the mainland. These funds will be used by the principal investigator to meet with JPL collaborators and/or attend relevant conferences (e.g., ICSE, FSE, ISERN, or STAR), where he may report on the results of his own work and obtain first hand information on related work.

### Materials and Supplies

The principal investigator directs the Collaborative Software Development Laboratory, which is currently equipped with one enterprise SUN server and 8 workstations running Windows2000, along with a printer and networking peripherals. The project budget provides for materials and services related to this equipment.

### Cost breakdown

All calculations are rounded to the nearest dollar.

**Summer Salary.** Summer salary is calculated as two additional months of the principal investigator's annual nine month salary.

**Research Assistants.** This cost category provides support for four graduate research assistants for each of the four years.

**Fringe benefits.** Fringe benefits for salaries are calculated at 3.5% for the principal investigator and 25% for the graduate assistants.

**Indirect costs.** University of Hawaii indirect (overhead) cost is calculated as 36.3% of Modified Total Direct Costs (MTDC), i.e. salaries, travel, and supplies (equipment is excluded).

# 8 Other matters

As requested by Michael Lowry, this cooperative agreement proposal contains the following attachments:

- The awarded proposal to the National Science Foundation HDCSSR program.
- A copy of the first year NSF report.