Enhancing Competitiveness via a Public Fault & Failure Data Repository[1]

Dolores R. Wallace
National Institute of Standards and Technology[2]
Information Technology Laboratory STOP 8970
Gaithersburg, MD 20899-8970 USA
dwallace@nist.gov Tel: 1-301-975-3340 Fax: 1-301-926-3696

1. INTRODUCTION

Today most software companies are under pressure to get reliable software products built quickly. Often managers of new products or new versions of released systems do not have time to conduct experiments on methods for software development or for preventing or detecting software faults or for reliability measurement. Yet, they are reluctant to buy into a method because of the difficulty of predicting how well it will succeed in prevention and detection. In other cases, managers may "know" their product is not reliable enough for release. Without measurement data from their project or a benchmark for their application class, they may be unable to convince upper management that the product will surely, and possibly embarrassingly or dangerously, fail when put into service. Data collection tools, and access to public data bases with tools to develop germane profiles and benchmarks, can help industry in deciding on methods, for assessing reliability of their products against others, and in identifying areas where improvement of their processes is needed.

Researchers who lack data from diverse projects have difficulty also in developing metrics about the effectiveness of methods toward achieving reliable software. Failures in high integrity systems are rare (and usually costly), and a single system usually does not accumulate enough data to permit meaningful statistical evaluations. Data from failures are needed for identifying the underlying causes of these failures and also in understanding the types of problems for which new methods are needed.

To assist both industry and researchers, the Information Technology Laboratory (ITL) at the United States National Institute of Standards and Technology (NIST) has initiated the Reference Data: Software Error, Fault , Failure Data Collection & Analysis Repository Project. Usually referred to as the EFF Project, the project treats the term "error" as the human action that produces the incorrect result; the term "fault" as the manifestation of an error in an artifact, and the term "failure" as the result of a fault that has been activated, or triggered, during operation of the software.

The EFF project is collecting and analyzing data from either the development and maintenance processes or the operation of a delivered computer system. The data, with all proprietary

---

[1]DISCLAIMER: Certain trade names and company products are mentioned in the text. In no case does such mention imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

information removed, will become available through a publicly accessible World Wide Web facility at NIST. The facility will include tools or Web links to tools to assist in developing profiles and benchmarks from the data. Researchers may use the data for their experimentation. The information technology industry and researchers may use the resulting reference data to assist in many types of decisions for building better end-user products and for developing new methods and tools to support industry. NIST encourages companies to consider the benefits of a public data base. NIST will accept new or existing private data to augment the repository. Such data may be from either the software development or maintenance process, or may consist of failure data derived from systems already in service.

The facility's analysis will enable users to construct various profiles, such as the frequency of fault classes within a an application domain or fault classes discovered by specific methods. By examining profiles of a previous version or of a similar product, project managers may avoid some problems via better staff training, improved checklists that address the most frequent faults (and guidance on how to use the checklists), and more accurate test scheduling.

2. DATA COLLECTION RESEARCH - LESSONS LEARNED

Research on software faults, and hence on data collection and analysis, has been ongoing almost as long as software development itself. One early paper [ENDR] asks several questions about errors for which the EFFTool is seeking data about faults. These questions, pertinent today, are shown in Table 2.1 and indicate that considerable information about an error is needed to learn from it. Such information includes data about discovery of the problem, description of the problem, its resolution, area or artifact of actual error cause, and the description of change.

RELEVANT QUESTIONS
Where was the error made?
When was the error made?
Who (generic) made the error?
What was done wrong?
Why was the particular error made?
What could have been done to prevent this error?
What detection method could detect it sooner?

Table 2.1 Questions for Software Error Analysis

Basili [BASI] proposed a highly organized approach for data collection and analysis, shown in Table 2.2. With respect to the EFF project, steps 1 and 2 are relatively easy. For step 3, some data categories have been easy to establish, but classifying both the symptoms that revealed errors and the cause of each error is difficult. Terms may be changed in the second version of the taxonomy. While a small group has reviewed and tested the concepts for the EFF project (Step 4), broader usage may require changes to some of the terminology. Validation of the data (Step 5) will be difficult because the data may be submitted in various media and formats from companies in varying locations. NIST will need to work with contributors to ensure understanding of the terminology used by NIST and by the contributors. Implementation of Step 6 relies on contributed data and availability of the data to researchers.

BASILI'S GUIDELINES	
1.	Establish the goals of the data collection
2.	Develop a list of questions of interest
3.	Establish data categories
4.	Design and test the data types
5.	Collect and validate data
6.	Analyze data

Table 2.2 Process for Data Collection and Analysis

Several companies plan to provide data from existing collections. Because such data will vary in content, data from these collections must be translated into the data categories evolving in this project. Special care will need to be exercised when analyzing data collected by a mechanism different from the NIST/ITL tool. In some cases, a WEB link may be provided to an existing data base of a company.

Fault and failure taxonomies and some models for descriptive fault and failure data are cited in Table 2.3. The diversity among these is great. Their research occurred in different domains, problem sizes, languages, and other variables. For example Endres' interest was in an operating system while Beizer [BEIZ] collected data from many projects of varying types and languages. And, much of the data that lead to these specific taxonomies was collected before the existence of modern languages (e.g., Ada, C++, JAVA) and widespread use of software tools aiding development. Web technology made it possible to incorporate a fairly complex taxonomy invisible to the user.

TAXONOMIES AND DATA MODELS		
Simple, few elements	Rubey Glass Weiss/Basili Grady Chillarege Fenton/Pfleeger	late 1970's 1981 1985 1992 1993 1996
Several groups, details	Endres Knuth	1975 1989
Security-oriented	Landwehr Aslam	1995 1995
CMU experiment: need data from complex projects	Greenberg/Siewiorek	1996

Maintenance	Stark	1997
Detailed, life cycle oriented	IEEE Standard 1044 Beizer	1993 1990

Table 2.3 Diversity Among Taxonomies

Questions arise. Do classifications apply to *all* languages equally? To *all* types of software? Has the advent of design tools, analysis tools, standards, and other parameters changed the nature of errors and hence the faults manifested in the artifacts? And, has the entry of more complex systems added failures that couldn't have been dreamed of before today's complex systems network usage? Unfortunately, because data are needed to develop new taxonomies and collecting data from many sources requires some predefined taxonomy, selecting the best fault and failure classification is very difficult. These issues need to be addressed when developing new taxonomies and characteristic profiles about faults and failures.

Key lessons learned from researchers and industry who have influenced the EFF project include the following:

- Attributes from large projects and small projects differ and will yield different fault frequencies.
- Normalization across projects needs to account for variability in project environments.
- Existing taxonomies have varying levels of details that make synchronization difficult.
- Fairly complex projects are needed for meaningful data.
- Data format must be kept simple.

3. THE EFF Project Toolset

Several Web tools are being developed by NIST/ITL to aid collection of the data and then to provide public access to the data:

- The EFFective Manager Tool (EFFTool), a Web software tool to aid collection and analysis of faults and failures found during the development or maintenance of software (released).
- The EFFSystem Tool, a Web tool for failure data from systems in operation (designed).
- The EFFPublicData Tool, a Web software system containing the data and some analytic and display tools and providing links to other repositories and statistical tools (in development).

Data contributed to NIST will be sanitized and then placed into the EFFPublicData Tool. The public data base will include data from the EFFTool and data contributed by other mechanisms. Whereas the EFFTool generates profiles only on fault and failure data, the public tool will use project data also in analyses. Including analyses of project data will enable further understanding of many projects regarding how faults and failures occurred, how they could have been prevented, and how they could have been detected earlier. These benefits primarily serve researchers, whether academia or the research department of a company. Direct benefits to industry are the profiles and benchmarks that can be constructed from the data to guide industry in choices about software development methods and in expectations for the reliability of their products. An example is indicated in the following story:

> You are leading a project with managers from other company divisions. You want to conduct software inspections on the code. Some managers do not believe inspections add value. To support your case, you access a public data base containing software fault and failure data to locate systems in your application domain with project characteristics similar to yours. You separate the systems into two groups: those who used software inspection and those who didn't. You normalize for project differences and next you build a profile for each group with their timelines which includes effort to fix problems, the fault count for inspection and different test activities. The results indicate whether inspections paid off in product quality, cost of the project, and delivery time.

The EFFPublicData Tool will be a WWW accessible data base system with access not only to data collected by NIST but to other repositories and to public domain analytic tools. Because the EFFPublicData Tool will contain data from many projects, across many application domains, other tools in the facility will assist in the problem of normalizing data from projects with differing attributes. Existing statistical and graphical tools are being explored for their feasibility for analyzing and displaying data and for direct linking from the NIST facility to them.

3.1 The EFFTool COLLECTION COMPONENT

The Web EFFTool collection component receives data by text entry, radio buttons, pull-down menus, and check boxes. The EFFTool collection component has two data forms: one for project description and one for each fault or failure. The data fields were selected from questions that researchers and organizations might ask. Obviously, foreseeing every question is impossible, but the data allow for many types of questions.

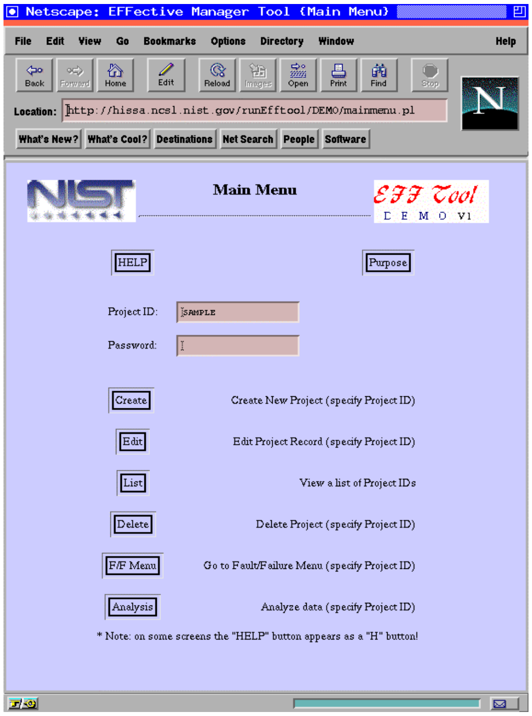
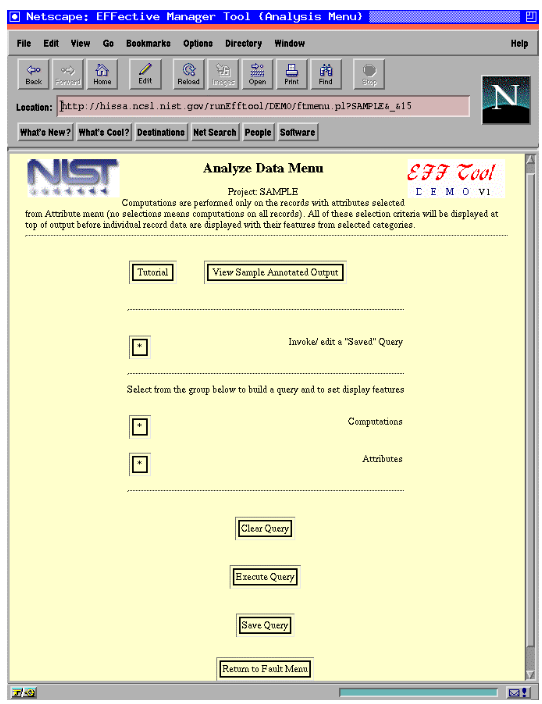


Figure 3.1 EFFTool Opening Menu

Figure 3.2 Analysis Component



The main menu shown in Figure 3.1 controls access to the project data forms, the fault/failure menu, and the analysis component (Figure 3.2). The EFFTool forms identify general project information and, for each fault or failure, specific information related to its discovery and resolution during the development or maintenance processes. Complete project management data, such as schedules, milestones, individual personnel skills and experience, are not included because many methods and tools exist to understand process itself.

Within an organization, environment features for projects may be the same and may be known to all who work on the project. These project features [Table 3.1] include elements such as development processes, the standards which govern the project, the programming language, and quality practices. For a single project, drawing conclusions about whether a specific method worked well may be relatively easy, provided other experimentation practices are followed [ZELK]. When data are collected across many projects and different organizations, these environment and organization elements are likely to differ. Researchers may combine data from like elements but they must be aware of differences and establish how they will treat the differences in their analyses.

TYPE OF PROJECT DATA	
Project name	CMM level
Company name & contact	Primary software language
Brief project description	Size of new & reused code
Date project began	% of COTS code
Development or maintenance	Requirements, design methods, automation
Perceived consequence of failure	Performer of QA / VV
Perceived criticality of failure	Company experience: in domain
Relation to hardware	Company experience: w/ software in domain
Generic domain	Company experience: w/ software in general
Specific application	Company quality practices
Contractual requirements	ISO 9000

Table 3.1 The EFFTool Project Information

The form for fault and failure data will be used for every fault or failure. A summary of the data categories for discovery is shown in Table 3.2. Basic questions like “when found, where found, who found” are asked, and a list of generic attributes is provided. The user has the option to specify more details on where a fault was discovered. For example, if the fault was discovered in the generic artifact "design," then the user may also enter a specific identifier name SUM_MONEY in a text field. The user supplies input on the possible consequence of this fault being allowed to remain in the system, and on the impact on the development schedule. The user assigns a priority to resolution of this fault. Assigning the priority may be a part of resolution in some organizations, or may be changed during resolution. The priority value, like data for any field, may be entered or changed at any time.

In fault discovery, a description of the discovery method, (e.g., inspections focused on initialization) may be entered. In a future version, this category may combine with Chillarege's triggers [CHIL]. Chillarege considers a trigger to be an activation process that activates a software failure from a fault, and the trigger is the focus of any investigative activities for faults. This is not quite the same as a method for discovering a fault or failure, but seems closely aligned. Chillarege has identified triggers for inspection, function test, and system test. It may be that both categories, discovery methods and trigger, will be useful for fault and failure analysis.

FAULT (and FAILURE) DISCOVERY	
Date of fault discovery	Potential severity if not fixed
Type of person who found fault	Impact on schedule
Specific location where fault discovered	Priority for resolution
Generic artifact where fault discovered	Discovery method
Activity during which fault discovered	Discovery method effort
Symptom	Degree method automated

Table 3.2 The EFFTool Discovery Data

Another data type presenting difficulty is the symptom of a problem. The symptom is the visible indication to the discoverer that something is wrong. Examples of symptoms include "ambiguity" and "execution stopped." When data are submitted to NIST, the "Other" field for the symptom category may indicate that the current list of symptoms needs to change. A similar problem occurs with fault classification in the resolution data, where the fault classification is the name of the actual cause. An example of the difference may be that an incorrect parameter is the visible symptom but the cause may have been a typing error or an incorrect specification. (And, the analyst needs to discover what caused the incorrect specification!) These two fields, like the trigger, are expected to iterate over time as more data arrive, especially from project using modern technology.

The resolution data types (Table 3.3) are similar to the discovery data in most data fields but there are differences. In resolution, the artifacts are those where the changes were made, which may lead to the actual source of the fault. For example, if the fault is found in the code, resolution may find that the source is in the design, and changes were made in both artifacts. The project manager will likely want to ensure that the error has been corrected, and that any other faults on other locations have been corrected. Researchers using the data to understand effectiveness of methods would in this case look for more errors in using the design method of this project and other projects using the same design method. One question asks if the fault occurred because of resolving some other problem. Tracking the answers to this question may lead to better understanding of the maintenance process.

One major challenge is the classification of faults and symptoms. The classifications must be easily understood and mutually exclusive. They need to apply to, or at least be understood with, all modern technologies. A long selection list may tax the patience of contributors. By selecting from pull-down menus for "artifact where discovered," "symptom," "artifacts changed," and "fault classification," the user is, without obvious effort, actually constructing a detailed taxonomy for that project. The problem is to synthesize terms from the many existing taxonomies. If usage of this tool indicates that contributors are willing to scroll through multi-level lists, then it may be feasible to go to more fine-grained fault classifications similar to Beizer's taxonomy [BEIZ]. The expectation is that data and comments from users of the EFFTool will be useful in defining a better taxonomy for second version.

FAULT (and FAILURE) RESOLUTION
Date resolution completed
Resolver (generic)
Classification
Generic artifacts fixed - also text fields for specific ids
Was this caused by a previous fix?
Project activity during which resolution is made
Resolution method
Degree method is automated
Resolution method effort

Table 3.3 The EFFTool Resolution Data

3.2 THE EFFTool ANALYSIS COMPONENT

To be competitive, companies need to get quality products out the door. To remain competitive, they need to learn from the current project and apply those lessons if possible during the current project and certainly to the next one. While other issues must be addressed for overall process improvement, the EFF project is concerned only with the collection and analysis of fault data.

The analysis component of the EFFTool aids the company using the tool in meeting its market objectives of timely delivery and improvement. The assignment to get the product out should translate to getting a *quality* product built and delivered *within budget and on time*. Because the term quality may have varying definitions, the assumption is that the project manager knows the organization's definition and has guidelines for judging acceptability of the product (Standard profiles of faults and failures for specific application domains could assist in defining acceptability and in identifying how to better the "standard" to beat competition.) Regardless of the definition of quality, the project manager needs to know the status of the faults and failures relative to their resolution and type.

The analysis component provides capability to select and count faults by status and other characteristics selected from the data elements and to perform various calculations and comparisons on those faults. The project (or test or quality assurance) manager should be able to use the analysis capability to monitor the project and to identify where to make changes related to methods, schedules, staff and other project concerns.

The fault or failure may have a priority for resolution and a person assigned to its resolution. The status may be either open or resolved where resolution of a fault or failure may result in any of these states:

- corrected,
- deferral of correction until another version of the program,
- correction resulting from a correction to another problem, and
- dismissal, that is, upon examination, no correction needed.

A user can ask for a count of all records with any combination of status conditions. Typical questions that the project manager may ask:

- How many faults with highest priority are open?
- What is the average number of days to resolve a fault?
- How many faults/ failures were discovered in a specific month (e.g., May 1997) ?
- How many faults/ failures were resolved in a specific month (e.g., May 1997)?
- What is the symptom of each open fault? What is its priority?

These general questions comprise a subset of many questions whose answers provide in-depth information about the project. Other information, such as the type of unresolved fault or the activity in which the fault was discovered, may enable decisions about methods for developing the software and activities to discover faults. A large number of faults relative to project size, schedule swings due to discovery of many faults and hence rework, and many faults not revealing themselves until they become failures in system test are among the many types of signals that a project needs some correction. The manager may not know these circumstances exist if data had not been collected and then reviewed. Once the data are reviewed, any number of actions may be taken, depending on the specific data. Examples include: reconsidering why inspections weren't conducted for the specific artifacts with the faults, or why a lower priority was assigned to some open faults that are obviously very serious? Studying the data may reveal that schedule requirements pushed the project into unit tests for those artifacts or insufficient traceability data lead to not assigning higher priority to those faults

Through a query system the analysis component provides the data to answer the types of questions cited in the previous paragraph. The analysis enables monitoring the status of open and resolved faults. The profiles derived from the analysis capability enable visibility to project managers and staff to establish priorities for resolving faults / failures, status of project as a whole, effectiveness of development or testing methods, training needs, technology needs, and other issues serving to improve current and future projects.

4. SUMMARY

NIST has initiated the EFF Project to create a public WWW facility containing software project fault and failure data to aid corporate management, project leaders of software systems or of software improvement groups. Tools supporting the analysis of the data will enable practitioners and researchers to view the data from various perspectives. For example, some may be interested in fault profiles of projects similar to their application to understand the value of certain methods to those applications. In another example, failure data from many systems in service may enable researchers to identify underlying causes of failure and hence attack the appropriate research problem to prevent those failures.

NIST is developing the EFF Toolset of WWW tools which will contain the EFFPublicData Tool for the public facility, the EFFective Manager Tool (EFFTool) for data during software development and maintenance, and the EFFSystem Tool for failure data from systems in service. The EFFPublicData Tool will house sanitized data contributed to NIST, provide links to other repositories and tools for analyzing and displaying data. The EFFTool contains features to assist project managers to track the fault and failure history on their projects and has been released to the public for their use. The EFFTool is intended to be used by companies at their sites, and its output is an ASCII file, for which NIST provides a script to translate the data to be acceptable to most spreadsheets, which usually have some graphics capability for the most common statistical measures. The tool may be downloaded from http://hissa.nist.gov/toolkit/eff.html.

5. REFERENCES

[BASI] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *TSE 10*, Number 6, November, 1984, 728-738.

[BEIZ] Boris Beizer, *Software Testing Techniques*, International Thomson Computer Press, 1990.

[CHIL] Ram Chillarege and Karen A. Bassin, "Software Triggers as a function of time - ODC on field faults," Fifth IFIP Conference on Dependable Computing for Critical Applications, IEEE Computer Society, 1995.

[ENDR] Albert Endres, "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering,* Vol.SE-1, No.2 June 1975, pp.140-149.

[ZELK] Zelkowitz, Marvin V., and Dolores R. Wallace, "Experimental Models for Software Diagnosis," NIST IR 5889, September, 1996