Data Collection

Importance of data

- Software measurement is as good as the data that are collected and analyzed.
- We cannot make good decisions with bad data.
- Let us look at what constitutes good data.
- Let us also discuss about issues related to collecting data on effort, size, duration and cost.

What is Good Data?

Your measurement program

- must specify not only what metrics to use,
- but what precision is required,
- what activities and time periods are to be associated with data collection,
- and what rules govern the data collection (such as whether a particular tool will be used to capture the data).

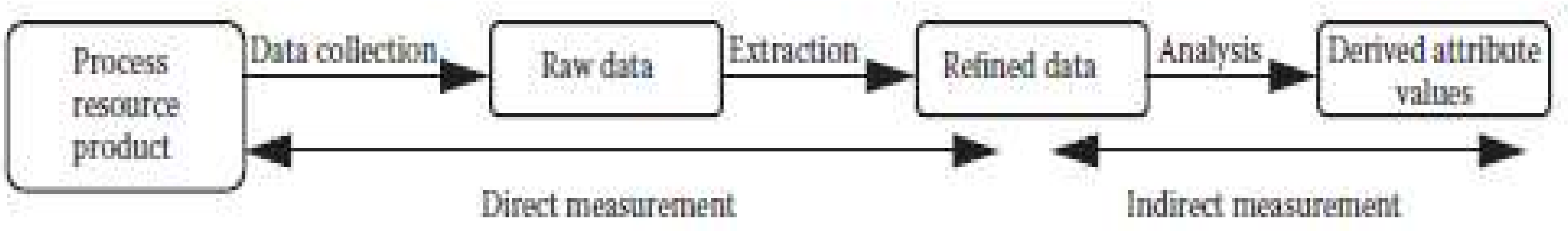
- **Are they consistent?** – Data should be consistent from one measuring device or person to another, without large differences in values.
- For example, two evaluators should calculate the same or similar values from the same requirement documents.

- **Are they associated with a particular activity or time period?** – The data should be time stamped so that we know exactly when they were collected. This can help us to track trends and compare activities.

- **Can they be replicated?** – The project histories and study results should be stored in a historical database so that baseline measures can be established and organizational goals set.

How to define the data?

- There are two kinds of data with which we are concerned: raw data results from the initial measurement or process, product or resource.
- But also *a refinement process, extracting essential data elements from the raw data so that analysts can derive values about attributes* as shown in figure below.



- *Consider the measurement of developer effort.*
- The raw data may consists of weekly timesheets for each staff member working on a project.
- To measure the effort expended on the design so far, we must select all relevant timesheets and add up the figures.
- We may also derive other indirect measures here such as average effort per staff member or effort per design component, etc.
- We must specify which direct measures are needed and also which indirect measures may be derived from the direct ones.
- Sometimes, we may begin with indirect measures.
- From GQM analysis, we understand which indirect measures we want to know; from those, we must determine which direct measures are needed to calculate them.
- Most organizations are different in terms of their business goals and corporate culture, development preferences, staff skills, etc.
- So GQM analysis of similar projects may result in different metrics at different companies.
- But most organization share similar problems – interested in software quality, cost and schedule (duration).

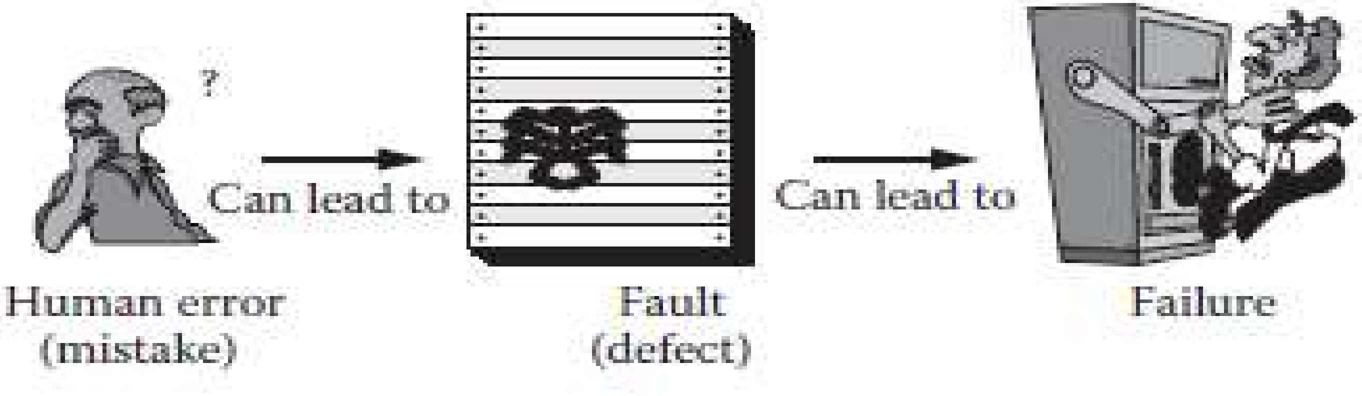
The Problem with problems

The developers measures aspects of software quality for following reasons:

- How many problems have been found with a product.
- How effective are the prevention, detection, and removal processes.
- Whether the product is ready for release to the next development stage or to the customer.
- How the current version of a product compares in quality with previous or competing versions.

Contd..

- The terminology used to support this investigation and analysis must be precise, allowing us to understand the causes as well as the effects of quality assessment and improvement efforts.
- However, the use of terms varies widely among software professionals, and terms such as “error,” “fault,” “failure,” and so forth are used inconsistently.



Contd..

- A fault occurs when a human error results in a mistake in some software product.
- That is, fault is the encoding of human error.
- The design fault can also result in incorrect code, as well as incorrect instructions in the user manual.
- A single error can result in one or more faults, and a fault can reside in any of the products of development.
- But failure is the departure of a system from its required behavior.

Contd..

- Failures can be discovered both and after system delivery, as they can occur in testing as well as in operation.
- Faults in the requirement documents can result in failures.
- We can think of faults and failures as inside and outside view of the system.
- Faults represent problems that the **developer sees**, while failures are problems that the **user sees**.
- Not every fault corresponds to a failure, since the conditions under which a fault results in the system failure may never be met.
- When undesirable or unexpected behavior occurs, we report it as a failure.
- The reliability of a software system is defined in terms of failures observed during operation, rather than in terms of faults.
- Systems containing many faults may be very reliable, because the conditions that trigger the faults may be very rare.

Terminology

- To many organizations, errors often mean faults.
- Anomalies usually mean a class of faults that are unlikely to cause failures in themselves but may eventually cause failures indirectly.
- Thus anomaly is a deviation from the usual, but it is not necessarily wrong.
- For example, deviations from accepted standards of good programming practice are often regarded as anomalies.
- Defects normally refer collectively to faults and failures.
- Sometimes a defect is a particular class of fault.
- Bugs refer to faults occurring in the code.
- Crashes are a special type of failure, where the system ceases to function.

- One fault can result in multiple changes to one product (such as changing several sections of a piece of code) or multiple changes to multiple products (such as change requirements, design, code and test plans).
- Following 8 attributes of a problem have been chosen to be mutually independent, so that proposed measurements one does not affect measurement of another.
  - **Location : where did the problem occur?**
  - **Timing : when did it occur?**
  - **Symptom : what was observed?**
  - **End result : which consequences resulted?**
  - **Mechanism : how did it occur?**
  - **Cause : why did it occur?**
  - **Severity : how much was the user affected?**
  - **Cost : how much did it cost?**

- Analysis of data may reveal, for example that severity of effect is correlated with location of fault within the product.
- However, this is something which is discovered after classification, and not something which is assumed when faults are classified.
- This characteristics of the attributes is called orthogonality.
- Orthogonality can also refer to a classification scheme within a particular category.
- One can consider following classification for faults, based on severity:
  - ✓ Major
  - ✓ Minor
  - ✓ Negligible
  - ✓ Documentation
  - ✓ Unknown
- This classification is not orthogonal.
- A documentation fault could also be a major problem, so there is more than one category in which the fault can be placed.

Failures

- A failure report focuses on the external problems of the system: the installation, the chain of events leading up the failure, the effect on the user or other systems, and the cost to the user as well as developer.
- Location is usually a code (say, hardware model and serial number, or site) that uniquely identifies the installation and platform on which the failure was observed.
- If the system is distributed, then the terminal at which a failure is observed must be identified, as well as the server to which it was online.
- Timing has two, equally important aspects: real time of occurrence (measured on interval scale) and execution time up to the occurrence of failure (measured on ratio scale).
- The symptom category explains what was observed, as distinct from the end result, which is a measure of the consequences.

- End result refers to the consequences of the failure.
- “End result” requires a (nominal scale) classification that depends on the type of system and application.
- End result of a failure may be any of the following:
  - ✓ *Operating system crash*
  - ✓ *Application program aborted*
  - ✓ *Service degraded*
  - ✓ *Loss of data*
  - ✓ *Wrong output*
  - ✓ *No output*
- Mechanism describes how the failure came about.
- This application dependent classification details the casual sequence leading from the activation of the source to the symptoms eventually observed.
- Unraveling the chain of events is part of diagnosis.

- Cause is also part of the diagnosis.
- Cause involves two aspects: ***the type of trigger*** and ***the type of source.***
- Trigger can be one of several things:
  - ✓ Physical hardware failure
  - ✓ Operating conditions
  - ✓ Malicious action
  - ✓ User error
  - ✓ Erroneous report
- Severity ***describes how serious the failure's end result was for the service required from the system.***

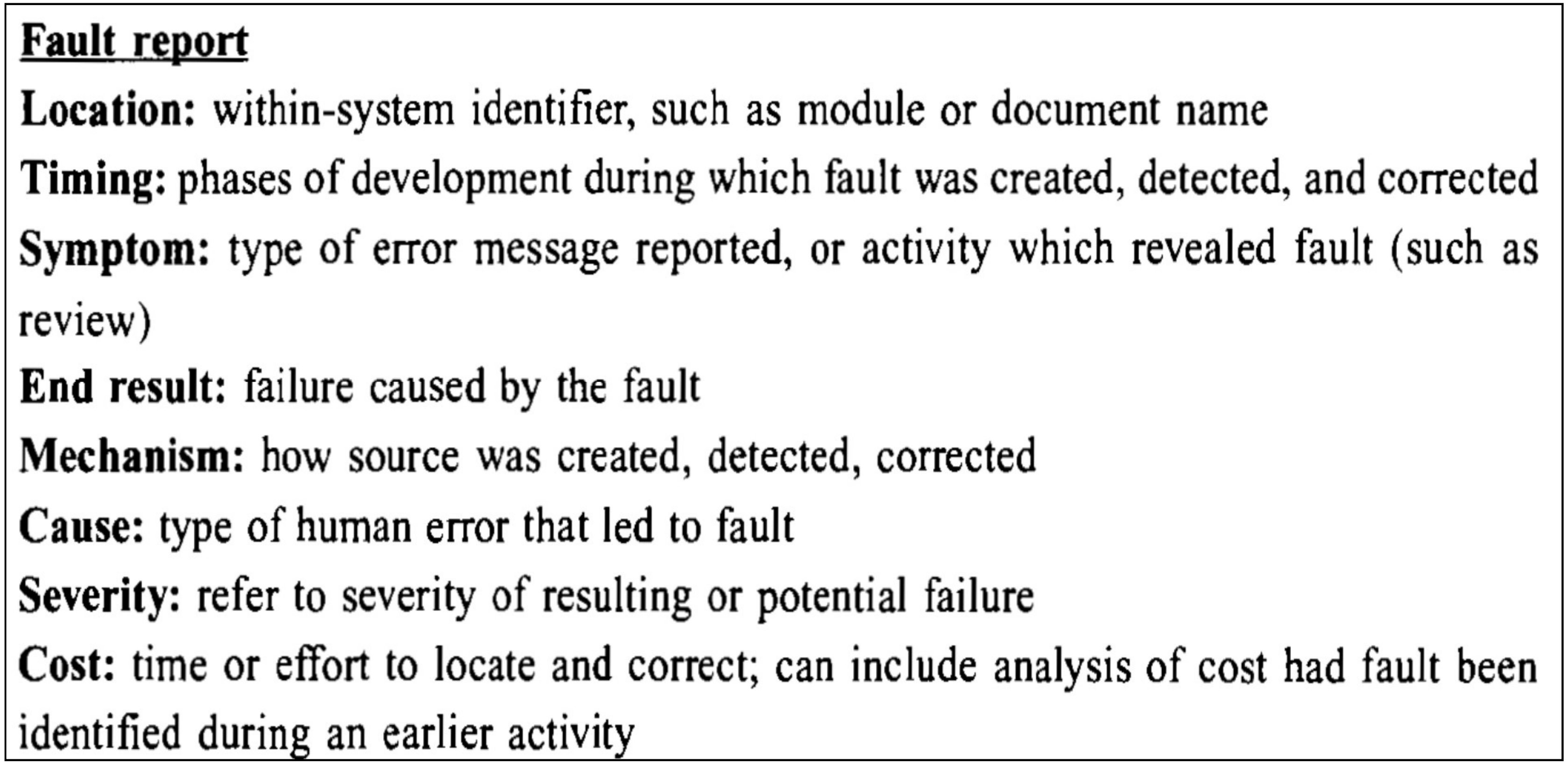
Failures in safety-critical systems are classified as:

- Catastrophic failures involve loss of one or more lives or injuries
- Critical failures cause serious permanent injury to a single person but would not result in loss of life to a person. It also includes failures causing environmental damage.
- Significant failures cause light injuries with no permanent effects.
- Minor failures result neither in personal injury nor in a reduction to the level of safety provided by the system.
- Severity may also be measured in terms of cost to the user.

- Cost to the system provider is recorded in terms of how much effort and other resources were needed to diagnose and respond to the failure.
- Sometimes, a failure occurs many times before it is recognized and recorded.
- Then a ninth category called, count captures the number of failures that occurred in a stated time interval.
- At times, a failure caused by user error might actually be due to a usability problem, requiring no immediate software fix but perhaps changes to the user manual.

Faults

- A failure reflects the user's view of the system, but a fault is seen only by the developer.
- Fault focuses on the internals of the system, looking at the particular module where the fault occurred and the cost to locate and fix it.
- A typical fault report interprets the 8 attributes:



Fault report

Location: within-system identifier, such as module or document name

Timing: phases of development during which fault was created, detected, and corrected

Symptom: type of error message reported, or activity which revealed fault (such as review)

End result: failure caused by the fault

Mechanism: how source was created, detected, corrected

Cause: type of human error that led to fault

Severity: refer to severity of resulting or potential failure

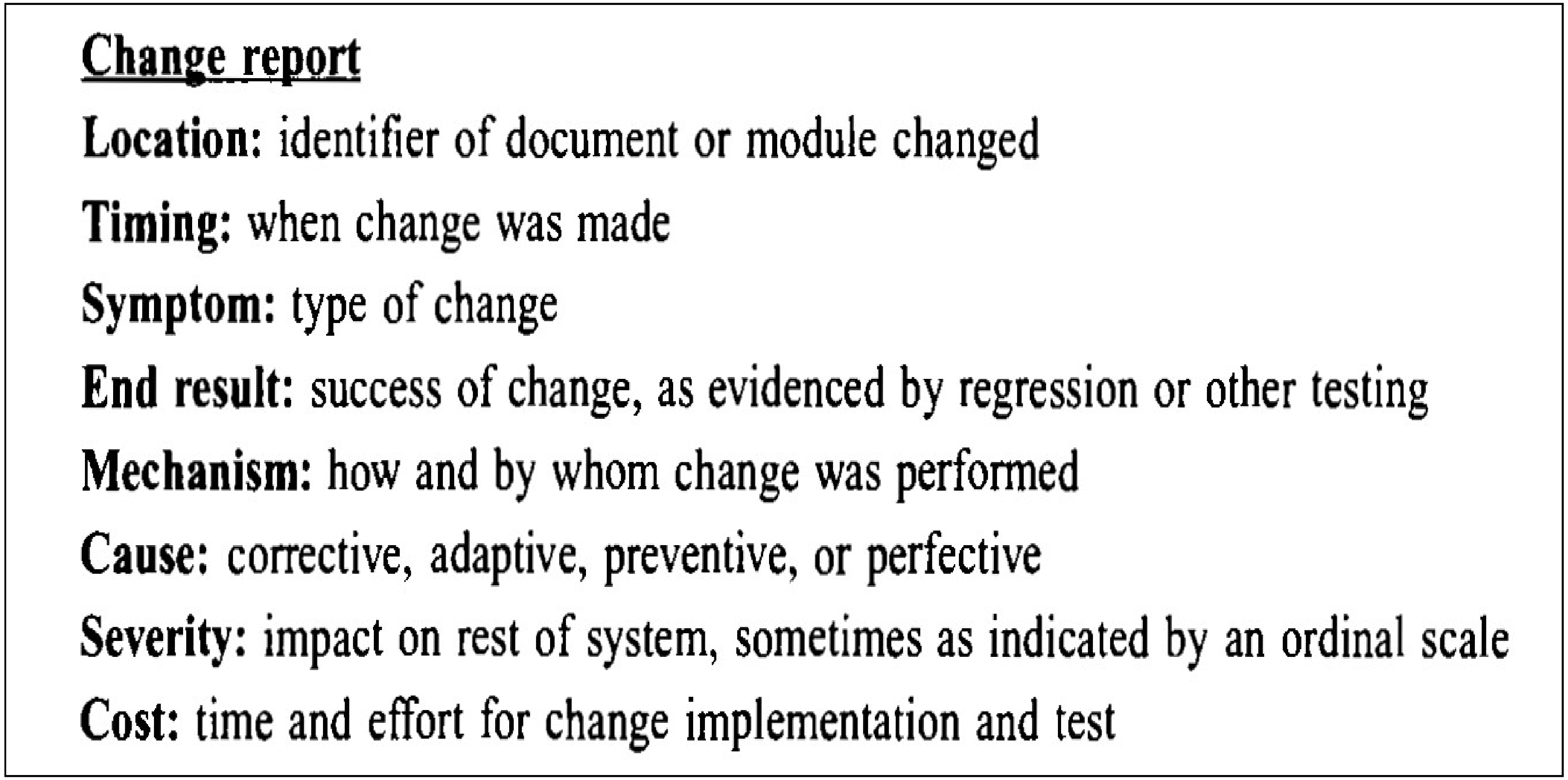
Cost: time or effort to locate and correct; can include analysis of cost had fault been identified during an earlier activity

- In a fault report, location tells us which product or part of the product contains the fault.
- Timing relates to the 3 events that define the life of a fault:
- When the fault is **created**, when the fault is **detected** and when the fault is **corrected**.
- The symptom classifies what is observed during diagnosis or inspection.
- The end result is the actual failure caused by the fault.
- Mechanism describes how the fault was created, detected and corrected.
  - Creation explains the type of activity that was being carried out when the fault was created.
  - Detection classifies the means by which the fault was found (eg, inspection, unit testing, system testing, integration testing)
  - Correction refers to the steps taken to remove the fault or prevent the fault from causing failures.

- Cause explains the human error that led to the fault.
- The cause may be described using :
- Communication: imperfect transfer of information
- Conceptual: misunderstanding
- Clerical: typographical or editing errors.
- Severity assesses the impact of the fault on the user.
- Cost explains the total cost of the fault to system provider. This can be computed only by considering other information about the system and its impact.
- Optional count can include several counts, depending on the purpose of the field.

Changes

- Once a failure is experienced and its cause determined, the problem is fixed through one or more changes.
- These changes may include modifications to any or all of the development products, including the specification, design code, test plans, test data and documentation.



Change report

Location: identifier of document or module changed

Timing: when change was made

Symptom: type of change

End result: success of change, as evidenced by regression or other testing

Mechanism: how and by whom change was performed

Cause: corrective, adaptive, preventive, or perfective

Severity: impact on rest of system, sometimes as indicated by an ordinal scale

Cost: time and effort for change implementation and test

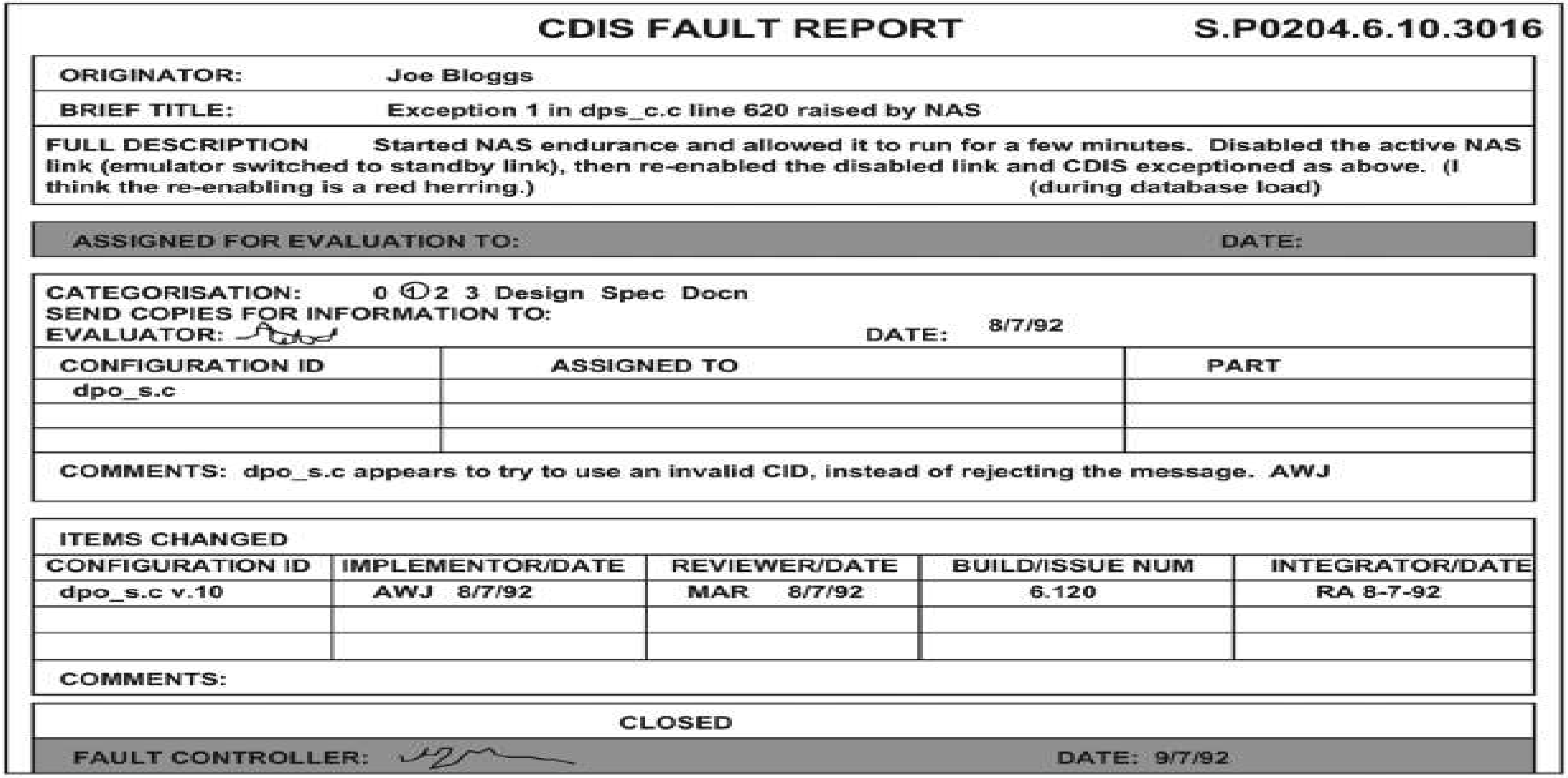
- The location identifies the product, subsystem, component, module or subroutine affected by a given change.
- Timing captures when the change was made, while end result describes whether the change was successful or not.
- Changes are made for one 4 reasons:
  - The change may be corrective, i.e it is correcting a fault that has been found in one of the software products.
  - It may be adaptive : the system changes in some way (ie when the hardware is changed, the given product must be adapted to preserve functionality and performance.
  - Changes also occur for preventive maintenance, when developers discover faults by combing the code to find faults before they become failures.
  - Developers sometimes make perfective changes, rewriting documentation to clarify the system structure

How to collect data?

- Manual recording is subject to bias, error, omission and delay.
- Automatic data capture is desirable and essential (eg recording the execution time of real-time software).
- Once the set of metrics is clear, and the set of components to be measured has been identified, we must devise a scheme for identifying each entity involved in the measurement process.
- We must make clear how we will denote products, versions, installations, failures, faults and more on our data collection forms.
- This step involves us to proceed to form design, including only the necessary and relevant information on each form.
- We must establish procedures for handling the forms, analyzing the data and reporting the results.
- One has to define who fills in what, when and where and describe how the completed forms are to be processed.

Data collection forms

- The data collection form encourages collecting good, useful data.
- The form should be self-explanatory as possible and should include the data required for analysis and feedback.
- It should allow the developer to record both fixed format data and free format comments and descriptions.
- Boxes and separators should be used to enforce formats of dates, identifiers and other standard values.
- A coded project identifier is used so that all forms relevant to one project can be collected and filed together.
- The name and organization of the person who completes each form must be identified so that queries can be referred back to the author in case of questions or comment.
- The date of form completion must be recorded and distinguished from the date of any failures or other significant events.



CDIS FAULT REPORT S.P0204.6.10.3016

ORIGINATOR: Joe Bloggs

BRIEF TITLE: Exception 1 in dps_c.c line 620 raised by NAS

FULL DESCRIPTION Started NAS endurance and allowed it to run for a few minutes. Disabled the active NAS link (emulator switched to standby link), then re-enabled the disabled link and CDIS exceptioned as above. (I think the re-enabling is a red herring.) (during database load)

ASSIGNED FOR EVALUATION TO: DATE:

CATEGORISATION: 0 [✓ 1] 2 3 Design Spec Docn
SEND COPIES FOR INFORMATION TO:
EVALUATOR: DATE: 8/7/92

CONFIGURATION ID	ASSIGNED TO	PART
dpo_s.c		

COMMENTS: dpo_s.c appears to try to use an invalid CID, instead of rejecting the message. AWJ

ITEMS CHANGED

CONFIGURATION ID	IMPLEMENTOR/DATE	REVIEWER/DATE	BUILD/ISSUE NUM	INTEGRATOR/DATE
dpo_s.c v.10	AWJ 8/7/92	MAR 8/7/92	6.120	RA 8-7-92

COMMENTS:

CLOSED

FAULT CONTROLLER: DATE: 9/7/92

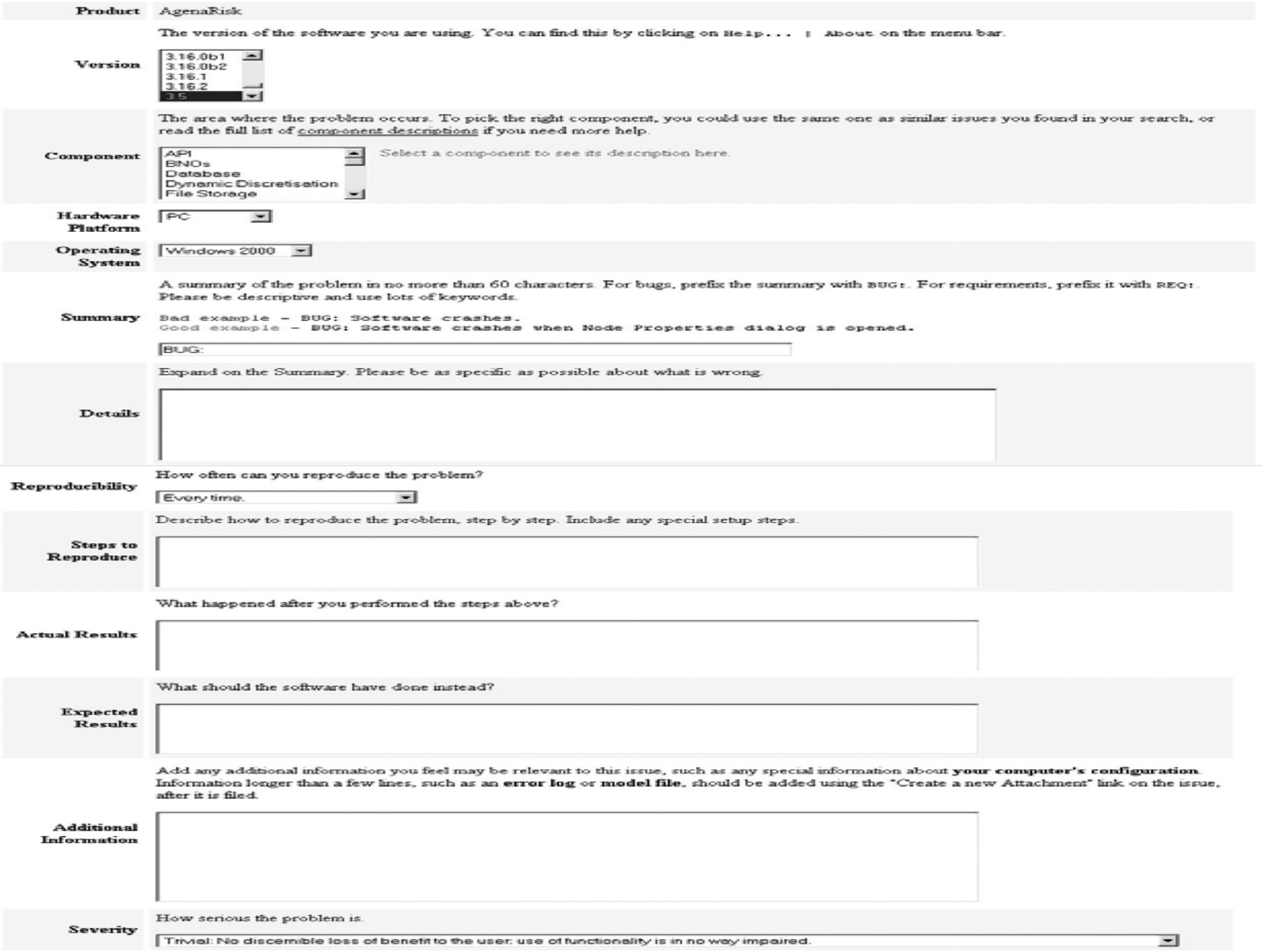
FIGURE 5.3 Problem report form used for air traffic control support system.

TABLE 5.3 Data Collection Forms for Software Reliability Evaluation

Identifier	Title
PVD	Product version
MOD	Module version
IND	Installation description
IRP	Incident report
FLT	Fault record
SSD	Subsystem version
DOD	Document issue
LGU	Log of product use
IRS	Incident response
CHR	Change record

Data Collection tools

- There are many software tools available that support the recording and tracking of software faults and their attributes.
- These tools provide data collection forms or frameworks for designing your own forms.
- There are 98 different commercial and freeware fault-tracking tools listed online.
- These tools can make it much easier for developers to monitor faults from their discovery to their resolution.
- The figure gives a screenshot of a tailored form using the **Bugzilla** open-source freeware tool.
- Tools to support data collection are constantly changing. One source for up-to-date information on data collection tools is the **International Software Benchmarking Standards Group (ISBSG).**



Product	AgenaRisk
Version	The version of the software you are using. You can find this by clicking on `Help...` | `About` on the menu bar. 3.16.0b1 3.16.0b2 3.16.1 3.16.2 3.5
Component	The area where the problem occurs. To pick the right component, you could use the same one as similar issues you found in your search, or read the full list of component descriptions if you need more help. API BNOs Database Dynamic Discretisation File Storage Select a component to see its description here.
Hardware Platform	PC
Operating System	Windows 2000

Summary	A summary of the problem in no more than 60 characters. For bugs, prefix the summary with `BUG:`. For requirements, prefix it with `REQ:`. Please be descriptive and use lots of keywords.
	`Bad example - BUG: Software crashes.`
	`Good example - BUG: Software crashes when Node Properties dialog is opened.`
	BUG:

Details	Expand on the Summary. Please be as specific as possible about what is wrong.
Reproducibility	How often can you reproduce the problem? Every time.
Steps to Reproduce	Describe how to reproduce the problem, step by step. Include any special setup steps.
Actual Results	What happened after you performed the steps above?
Expected Results	What should the software have done instead?
Additional Information	Add any additional information you feel may be relevant to this issue, such as any special information about **your computer's configuration**. Information longer than a few lines, such as an **error log** or **model file**, should be added using the "Create a new Attachment" link on the issue, after it is filed.
Severity	How serious the problem is. Trivial: No discernible loss of benefit to the user; use of functionality is in no way impaired.

When to collect data?

- The actual data collection takes place during many phases of development.

- For example, some data relating to project personnel can be collected at the start of the project (eg qualifications or experience) while other data collection, such as effort, begins at project start and continues through operation and maintenance.

- The counts of number of specification and design faults in various intermediate products and product components can be collected as inspections are performed.

- Data about changes made to enhance the product, as opposed to correct faults, can be collected as enhancements are performed.

How to store and extract data?

- Raw software engineering data should be stored on a database, set up using a DBMS.
- An automated tool for organizing, storing and retrieving data, a DBMS has many advantages over both paper records and "flat files".
- Languages are available to define the data structure, insert, modify and delete data and extract refined data.
- Constraints such as checks on cross references among records, can be defined to ensure consistency of data.
- Formats, ranges, valid values and more can be checked automatically as they are input.