



# Subjective software evaluation

R. Dick & R. Hunter

*Department of Computer Science, University of Strathclyde, Glasgow, UK*

## ABSTRACT

The distinction is often made between objective and subjective software measurements, objective measurements being preferred in many cases since they are more likely to be repeatable and reproducible. It is argued, however, that complex artifacts such as software products, cannot be evaluated purely from objective measurements and that expert judgement also has a role to play. A tool which supports the software evaluator by integrating a source-code browser with a check-list manager is described, and it is concluded that software evaluation, like software development, is best seen as a symbiosis of human and machine, each performing the tasks to which it is best suited .

## INTRODUCTION

By their very nature software products are not readily evaluated. There are no visible characteristics to give any clue to the quality of the product, and the behaviour of software systems is often complex and non-continuous, in that small changes to the data input to a system may substantially change its output.

The international standard ISO 9126 [1] has identified the quality factors relevant to software to be Portability, Efficiency, Reliability, Functionality, Usability and Maintainability. To some the most significant quality factor, and the principal one which will be discussed in this paper, is Functionality which relates to the extent to which the Requirements of the software product have been successfully implemented. The most obvious, and the most direct method of evaluating this quality factor is by testing the software ie. executing the software with appropriate test data. Notice that the term *testing* when referred to software, takes on a rather restricted meaning compared to its meaning when applied to other products. Testing is more generally taken to mean any activity which leads to information concerning the quality of the product concerned, and we may at times use this broader meaning of testing in connection with software as well, when no confusion is likely to arise.

Software testing, in the narrower sense, may be used to discover faults in the software and testing methods may be rated on their abilities to find faults. A lack of failures during testing may also be used to infer a lack of faults in the software. However, any realistic piece of software cannot be exhaustively tested and, because of this, other methods of evaluating software have been investigated, the principal ones being

measurement of the software product in some way

and

evaluation of the process used to develop and maintain the software product

Much work on software process evaluation has taken place in recent years based on the international (generic) *process* standard ISO 9000 [2] and the Software Engineering Institute's Capability Maturity Hierarchy [3] and other similar schemes. Software Process Evaluation, however, is

beyond the scope of this paper and will not be considered further.

The ESPRIT project SCOPE [4], in which the University of Strathclyde was involved, and which terminated in June 1993, was concerned with software product evaluation and developed a number of independent evaluation modules to evaluate software in terms of the six quality factors of ISO 9126. These evaluation modules were based on a mixture of software testing (in the narrower sense), software measurement and expert judgement based on standardised questionnaires. SCOPE monitored 27 Case Studies in about 20 European countries, each involving a software developer in close touch with a member of the SCOPE consortium. Product (and in some cases *process*) data was collected by the SCOPE partner involved and analysed locally as well as (in many cases) being transferred to the SCOPE central database held at the University of Strathclyde in Glasgow. By the end of SCOPE the database held in excess of 10000 records of up to 41 fields each, containing software product metrics, principally static analysis and reliability data.

SCOPE experimented with and evaluated a number of evaluation modules for each of the Software Quality Factors defined by ISO 9126, and four levels of attainment (A to D) were identified for each of the factors. Where appropriate the evaluation modules were supported by software tools such as static analysers and check-list managers. In this paper we will discuss the relative values, as we see them, of the various approaches to software evaluation mentioned above. The opinions offered are the authors' alone and do not necessarily reflect the views of the former members of the SCOPE consortium. We will also describe a tool designed to support the human software evaluator built by one of us (RD).

## OBJECTIVE AND SUBJECTIVE SOFTWARE EVALUATION

It is often argued that evaluation of software products based on objective measurements is much to be preferred

## 324 Software Quality Management

to evaluation based on (subjective) expert opinion, in that the former method is *repeatable* in the sense that a later evaluation by the same evaluator will produce the same result, and *reproducible* in that the same result may be obtained by another evaluator using the same criteria, whereas the latter method may not be. This is, of course particularly important in the context of the European Community where software evaluations performed by one organisation in one of the Member States should be recognisable throughout the Community.

Purely objective evaluation by means of metrics, however, tends to lead to rather arbitrary limits being set for the allowable ranges of the metrics being measured, and it is very difficult in most cases to understand the justification for setting the particular limits used. Some metrication tools even have arbitrary limits built in to them which the user may vary to suit local circumstances. In some well understood environments using mature software processes, the user may be able to choose appropriate limits from experience of past projects, though in most cases, we feel, it would not be at all easy to choose them. When it comes to independent software evaluation by a third party it is doubtful whether consideration of the metric values *alone* would be of great value to the evaluator, though the values may be useful in connection with other information available to the the evaluator, or in helping to identify what further information he requires to look at in evaluating a software product.

A simple illustration may be used to illustrate the last point. Last year a static metric evaluation tool [5] was applied to some source code produced by student projects at the University of Strathclyde. The metric tool allowed the user to browse through various tables of metric values and graphs showing the control and call graphs of the source code etc. It was not difficult to find out that in some cases recursive procedures had been used. Since there are circumstances in which recursion is considered to be poor programming practice, it was decided to examine the source code corresponding each of the instances of

recursion which had been identified. In most cases the recursion was seen to be associated with the application of standard algorithms to recursive defined data structures which, in the environment in which the students were working at least, was considered to be good programming practice. In a few cases however the recursion was seen to be associated with the validation of input data which was performed not by an iterative loop but by calling the input function within itself when invalid data was read. This was not considered to be good programming practice but would have been very difficult to distinguish from other acceptable uses of recursion purely objectively and without the use of 'expert opinion'.

The issues of repeatability and reproducibility are the main reasons why expert judgement is not readily accepted as a means of evaluating software. In this sense (though perhaps in no other) software may be compared with wine. Expert wine tasters would not necessarily agree on the merits of a particular wine, though they would probably tend to agree in the majority of cases. Of course objective tests of the acidity, alcohol level, amount of tannin etc would be more repeatable and reproducible than expert judgement, though many would feel such tests to be of much less value. In the case of software it may be the immaturity of the subject that causes expert opinion to differ, and the insistence that only objective measures should be considered in software evaluation, serves only to hide the lack of agreement among experts as to what measurable quantities relate to software quality.

Of course objective measurements and subjective judgements should not be seen as alternatives. In many cases subjective judgements based on objective measurements will be appropriate. There is little point in expecting an expert to do what can be done much more quickly and more reliably by a machine. Machines are good at doing highly repetitive tasks, computationally intensive tasks and other tasks which a human would find boring and time consuming. However the objective measurements produced by tools are of maximum value in

## 326 Software Quality Management

the hands of an expert who may use them as he sees fit. In particular he may compare which values he chooses, reject values, seek further information and so on.

Once the need for expert judgement in evaluating software is accepted, tools (ideally an evaluation workbench) should be made available to support the work of the expert, so that maximum value may be obtained from his expertise. The following types of tools should prove useful

- browser tools
- tools to access historical information from other projects
- analysis (statistical and otherwise) tools

The tools should be integrated where possible and should offer clear and easily usable user interfaces. In the next section we describe a browser tool which was integrated with a questionnaire manager to provide an environment for an expert evaluating source code with the aid of a check-list.

### THE BROWSER/CHECK-LIST MANAGER

#### The user interface

Design of the user interface was considered to be an important part of the process involved in developing the tool. The major design goals were

- *encapsulated functionality*. The user interface will be the only way the user can interact with the underlying system. It is essential therefore that the full functionality of the system is available through the user interface.

- *user friendliness* Users of the system must find it aesthetically pleasing and easy to use.

- *customisation* Users of the system should be able to change the appearance of the user interface to suit their own taste.

Before arriving at its final appearance the interface went through a number of *prototype* stages. Each prototype was evaluated according to the above design aims. User friendliness, in particular, could not be evaluated objectively, and the views of 'representative' users was sought at each stage.

Tools are available for building user interfaces and, in this case, the X Toolkit for the X-windows [6,7] system was used. The basic data type and fundamental abstraction of the X toolkit is the *widget*. Widgets are user-interface components, each with a class which determines how it reacts to user interactions via the mouse and keyboard. An *instance* of a widget class - the widget itself - has attributes which govern its behaviour and appearance.

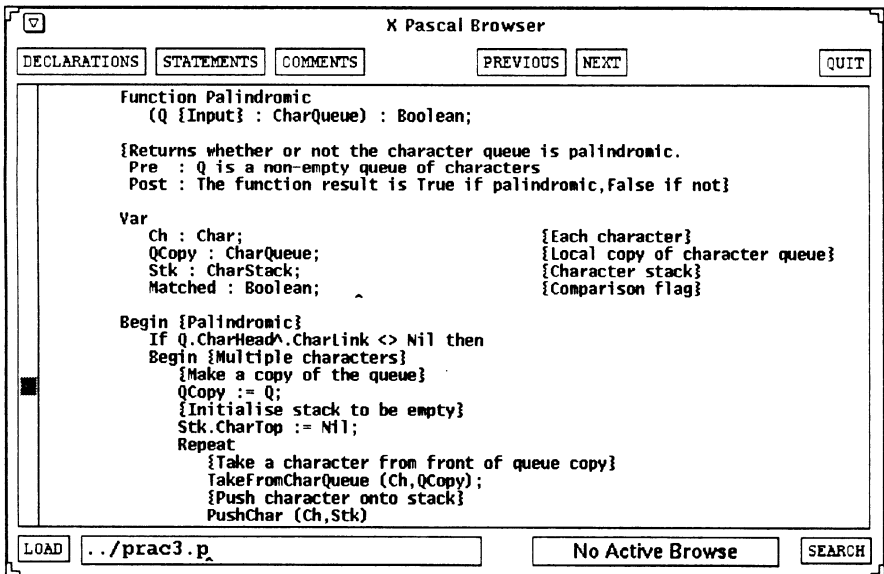


figure 1

## 328 Software Quality Management

For example, a text widget is a widget class which allows text to be displayed, edited and scrolled within a window. However, an instance of the text widget class could be specified as being read only. The X toolkit provides functions for creating and destroying widgets and for setting and retrieving widget attributes.

### The browser

The browser is a syntax-directed source-code browser for Standard Pascal source code developed with the aid of the Unix compiler-building tools Lex and YACC; Lex being a lexical analyser generator and YACC (Yet Another Compiler-Compiler) being a bottom up LALR parser generator.

The browser is able to locate sets of type definitions, variable declarations, while statements etc. and allows the user to browse through them. Figure 1 shows the environment in which the user works. The browser will only accept files with correct syntax and will, if appropriate, display an error message relating to the first fault found, in a popup window which appears in the middle of the screen as shown in figure 2. The browser, however, only detects context-free syntax errors, and errors such as variables which are undeclared, or inconsistent *types* in an *assignment* will go undetected.

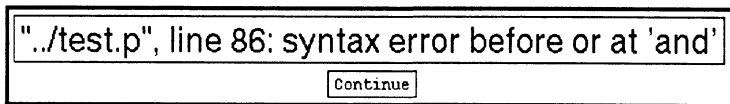


figure 2

The browser may be used in several ways

- the arrow keys on the keyboard can be used to move the cursor around the text
- the scrollbar to the left of the text may be used to scroll the text up or down
- various types of declarations and statements may be located by suitably selecting from pull down menus as shown in figure 3
- the buttons *previous* and *next* may be used to locate the next instance of a type of declaration or statement selected as above
- the command button named *comments* may be used to locate comments in the programme

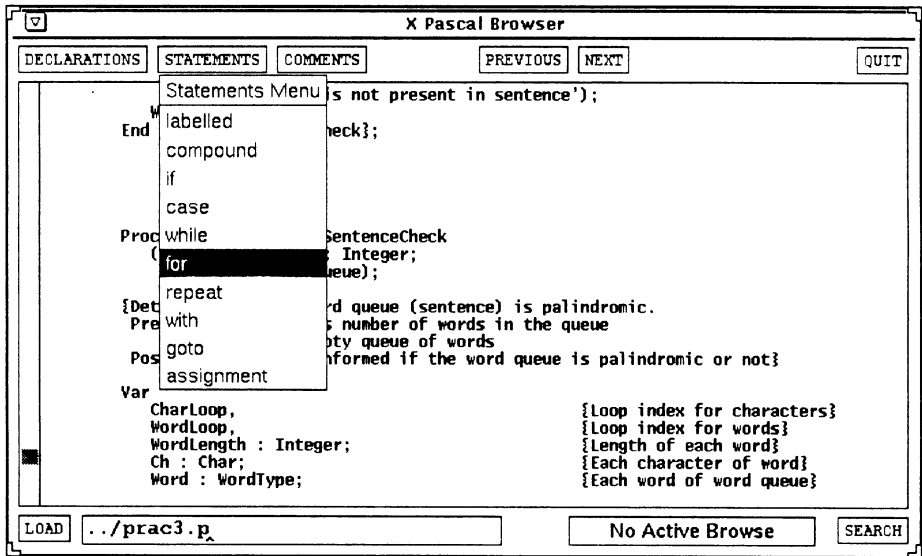


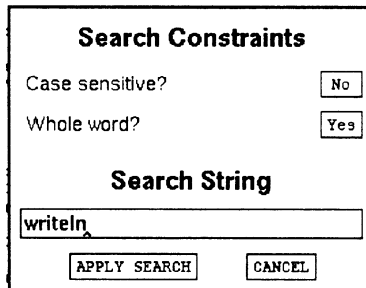
figure 3

### 330 Software Quality Management

- arbitrarily selected strings may also be searched for. The search may be case sensitive or case insensitive and may involve matching whole words or not (see figure 4).

#### The Questionnaire Manager

The Questionnaire Manager also runs under X Windows, is mouse driven and has a graphical interface. It allows arbitrary multiple-choice questionnaires to be presented to the user and is responsible for managing the responses (see figure 5). A questionnaire consists of a *title* and *questions* for each of which there are one or more possible *answers*. A sequence of one or more questions may be associated with a single *topic*.



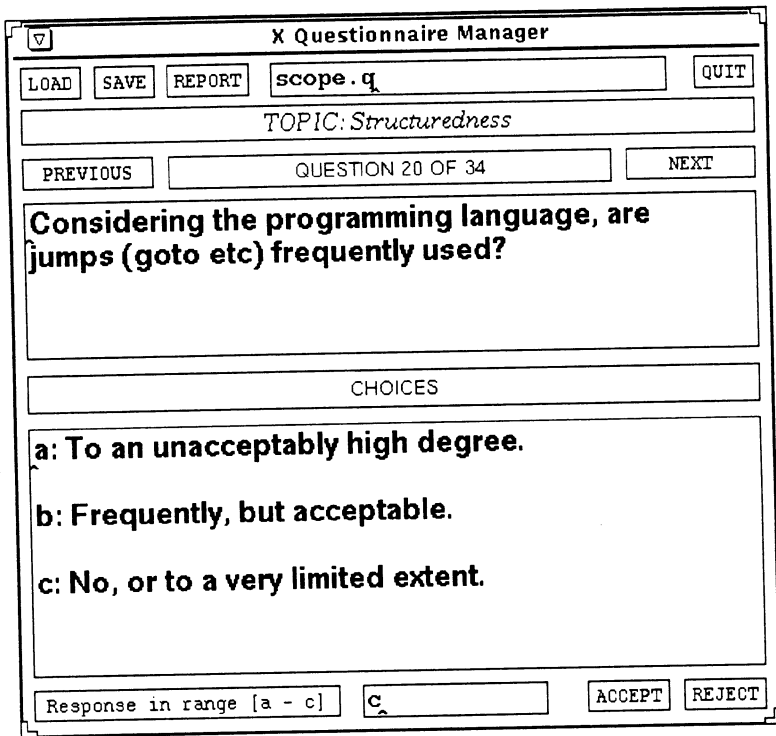
**Search Constraints**

Case sensitive?

Whole word?

**Search String**

figure 4



The screenshot shows a window titled "X Questionnaire Manager". At the top, there are buttons for "LOAD", "SAVE", "REPORT", and "QUIT". A text field contains "scope.q". Below this, a label "TOPIC: Structuredness" is displayed. A navigation bar includes "PREVIOUS", "QUESTION 20 OF 34", and "NEXT". The main question area contains the text: "Considering the programming language, are jumps (goto etc) frequently used?". Below the question is a section labeled "CHOICES" containing three options: "a: To an unacceptably high degree.", "b: Frequently, but acceptable.", and "c: No, or to a very limited extent.". At the bottom, there is a response range indicator "Response in range [a - c]", a text field with "c", and buttons for "ACCEPT" and "REJECT".

figure 5

Questionnaires are represented as consisting of titles, topics, questions and answers and may be specified as follows

```
%title string [(%topic string) %question string (%answer string)+] +
```

where *string* in italics stands for an arbitrary string, the topic part is optional and + stands for one or more occurrences of what precedes it. The strings preceded by '%' stand for themselves. The syntax specified allows questionnaires to be specified simply and efficiently while at the same time permitting changes with minimal effort.

## 332 Software Quality Management

To load a questionnaire from a file the user must type the questionnaire's name into the file name text *widget* to the right of the REPORT button and then click the command button marked LOAD. The syntax of the file will then be checked and if a fault is found, a window describing the fault will appear in the middle of the screen with a message such as

"test.q", line 1:"%title" expected

If the syntax of the file is correct the questionnaire will be loaded and the first question will be displayed. Widgets are used to display the topic of the question, the question number and the range of possible responses (eg a-c). The question and the possible answers are displayed in read only text widgets with scroll bars, if there is not room to display all the text at once.

To answer a question the user types the letter attached to the answer they choose into the response text widget at the foot of the screen in the middle, and clicks the ACCEPT command button. The response will be recorded and the next question will be displayed along with the possible answers. The user can delete the response to a question at any time by clicking on the REJECT button when the question is in the window.

Questions may be answered in any order and not necessarily in numerical sequence. Two command buttons PREVIOUS and NEXT may be used to browse through the questions. This facility allows users to inspect the questionnaire before answering any of the questions. When all the questions have been answered users may do one of two things. Firstly they may choose to save the responses in a file by typing the file name into the filename text widget and clicking on the SAVE command button. Alternatively they may cause a questionnaire report to be written containing all the questions and answers together with the chosen response to each question. Topic headings will also appear centred on the page and additional spaces between words will be

suppressed. To do this users will type in the name of the file in which the report will appear in the filename widget, and click the REPORT button. In recognition of the fact that the report is likely to be viewed on a terminal or printed on a printer, it is produced in long lines word wrapping over an 80-character border.

## CONCLUSIONS

As already mentioned, the tool described is only one of a number of tools which might be produced in order to assist the human evaluator. Use of such tools should serve to optimise the efficiency and reliability of the human evaluator by providing a pleasant environment to work in and removing as much routine administration from him or her as possible. From the experiences described we believe that Software evaluation, as with software development, should, involve a symbiosis of man and machine each performing the aspects of the task to which it is best suited

## REFERENCES

1. ISO 9126, *Software Product Evaluation - Quality Characteristics and Guidelines for their use* international Standards Organisation, 1991.
2. ISO 9000, *Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing*, International Standards Organisation. 1987.
3. Humphrey WS, *Managing the Software Process*, Addison Wesley, 1989.
4. SCOPE Project, *SCOPE 'Leaflet'*, ESPRIT project 2151, Prime contractor VERILOG of Toulouse, 1992.
5. *Prometrix, Static analysis and metric evaluation tool*, developed by Infometrix Software, Glasgow, 1993.



### 334 Software Quality Management

6. McCormack J, Asente P, Swick R, *X Windows System Toolkit: the complete programmers guide and specification*, Digital Press, 1990.
7. Young D A, *X Window Systems Programming and Applications with Xt*, Prentice Hall, 1989.