

# Software Quality in User Centred Design

Massimo Felici, Alberto Pasquini, Stefano De Panfilis

## Abstract

*The paper describes a practical experience of measuring and modelling software quality using a development process based on the User-Centred Design approach. A crucial aspect of this approach is related to the evolution of the developed system due to the iterative design process. The experiment concerns of designing and evaluating prototypes for a train traffic control system for the Italian National Railways (FS). The paper points out the critical issues in a measurement practice starting from the early phases of the project. The description of the quality model for the project is presented. Moreover, the criteria identified to select the metrics and the metrics used are described together with the problems encountered in relation with this specific design approach. The paper points out the need for more suitable and powerful metrics and suggests in which direction the metrics should be up-dated. The reported experience shows that use of similar metric sets should be encouraged, but comparison of measurement values must be limited to projects with similar characteristics.*

## 1. Introduction

The practice of measuring quality is becoming a very important part within the development of a software product. Over the last decade, there has been a substantial rise in the research on and practice of software quality assurance [8], but there is still confusion about how to organise a quality plan based on measures. The engineering practice for planing a quality metrics program is not yet completely defined [2, 3, 7]. There is an increasing number of software process and product standards that emphasise the need for measurement [9, 10, 11, 12]. For example, ISO 9001 states that the process control shall include monitoring and control of suitable process and product characteristics during production and installation. However, although the requirements to measure product and process from the viewpoint of quality are stated, there is little guidance as to what should be measured and how the measures can be used to support the development of high quality software. An additional problem is that measurements cannot be defined in a totally context independent manner. In fact, the definition of quality in a quantitative way passes through a quality model strictly related to the (kind of) project and the type of application [1].

The paper is organised as follow. Section 2 shows the adopted development process in the context of the implementation of the control system for the Italian National Railways (FS). In section 3, we introduce the quality model defined for the control system with an instance of the quality requirements. In defining a quality model there are several critical issues that are described in section 4, which contains also selection criteria for metrics. Finally, section 5 shows the policies which will drive the close to start data collection activity.

## 2. System description

The project aims to design and prototype a control system for rail traffic in co-operation with the Italian railways operator (FS) as potential user. This system supports the dynamic rescheduling of the rail resources (train paths, vehicles and maintenance units) by human agents in case of unplanned external events. The planning activity as currently conducted by FS is divided into seasonal time-table planning, contingency planning to take into account

special short term demands, and real-time planning to react to unforeseen events. The project focuses on real-time planning on a particular line of the Italian railways network which can be characterised by the need for high responsiveness, rapid decision taking and the existence of multiple conflicting targets. Important requirements for the system to be developed during the project are:

- increased quality of service (e.g., rescheduling the rail resources via real-time planning in the case of disturbances, providing customer information);
- more efficient use of infrastructure, personnel and rolling stock;
- dynamic management of transport resources.

The project is based on the User-Centred Design approach. The main reason is that the project aims not only to develop a control system, but also to design the work performed from the operator that uses the system. Task analysis takes into account the user viewpoint in order to obtain a wide accepted system. The user suggests/guides the system features through the task analysis results. The more specific phases of this approach are: design based on Task Analysis; prototyping; early evaluation of the system functions with the user; and again to Task Analysis.

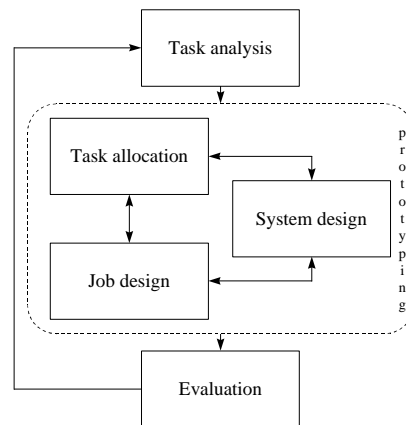


Figure 1. Phases of the design process.

Figure 1 shows the phases of the design process. During the task analysis phase, the project staff studied the problems involved in different domains (task world, user knowledge and behaviour, organisational factors, existing support mechanisms). The knowledge gained is structured in a formal representation mapping the process as it is currently carried out and its related objects and entities to the model world. Task allocation and job design work in parallel since the aim is to derive an optimal overall allocation of tasks between human and computer. Several architectural and implementative solutions are evaluated with the user by refining the prototypes and repeating the task analysis in an iterative way. This phase takes into account directly measurable criteria such as performance (e.g., number of trains dispatched, maintenance work completed) and also criteria such as usability, cognitive workload, and level of cognitive support. They are evaluated by video-taping users working through scenarios, interviews, questionnaires, and reports. This is useful to collect data, highlight general and particular problem areas and to provide feedback to the next iteration of the design cycle.

Figure 2 shows the general software architecture of the control system. There are three different main components (four with the operator), the System Monitor, the Man Machine Interface, and the Decision Support System.

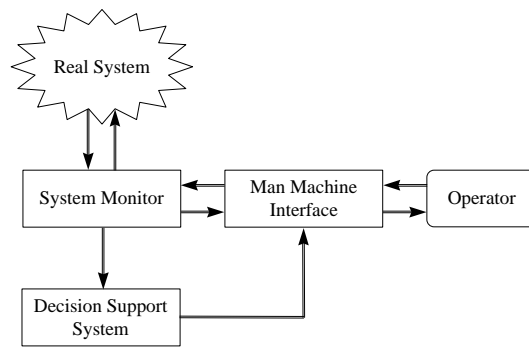


Figure 2. Scheme of the control system.

Each of the above components has a different task and then different quality requirements. For example, the functions assigned to the Man Machine Interface and System Monitor require that these components have a high reliability level. The operator controls/monitors the real system through these two components. Any failure of these two components affects the main system functions decreasing the traffic control abilities of the operator. The Decision Support System is the intelligent part of the system. It suggests solutions and predicts system behaviours to the operator. It is very important for the whole system, but it is not essential. The final choice for controlling traffic is still a human decision assigned to the operator. If you exclude the decision support system, the operator is still able to control the system, but with reduced performances because of the lack of automatic support. During early stage of the project, we conducted an analysis of the quality requirements for each component in order to identify possible conflicting requirements. Table 1 shows the quality requirements for the control system. They are defined upon a nominal scale of four levels (Very High, High, Medium, Low).

### 3. Modelling quality

One of the activities in the software development life cycle is the evaluation of the product quality achieved. The evaluation is based on the quality model specified for the specific (kind of) project. The quality model consists of a set of important quality characteristics for the final product. During the first phases of the project, we defined a quality model specified in terms of quality characteristics and sub characteristics. Quality sub characteristics are refined until they are referred to as quality attributes. Attributes are directly measurable software properties that quantify quality sub characteristics. Quality characteristics and sub characteristics can be internal or external. Properties that relate to the quality characteristics of the final product as seen by its user are called external quality characteristics (typically obtained by measuring the behaviour of the code when executed). Properties related to the way the product was developed and which therefore influence the quality characteristics of the final product are called internal quality characteristics (typically obtained by static measures of the code). The main goal to adopt a measurement practice is monitoring project progress. The spirit of the distinction between internal and external is quite intuitive. You should obtain a product of recognisable quality, if it is “well developed”. In fact, internal quality attributes are measurable properties of a software product on its development process that influence (or are believed to influence) final product quality. They are related to one or more external quality characteristics of the quality model. During project planning you need to set targets on internal quality attributes in order to monitor the

project progress. Once a product is undergoing final testing or has been released to a limited user population, it is necessary to measure the actual quality achievement. You need to measure your external quality attributes and compare them with their targets to confirm that you have delivered a product that meets its requirements. These requirements, usually referred as quality requirements, describe the user needs and are quantified in terms of values (targets) for specific external attributes. These values are related to the requirement levels (Table 1).

In order to identify the quality characteristics and to define the quality requirements for the project, the task analysis is quite useful. Task analysis results allow to better understand the integration of the system into the work environment. It is possible to analyse deeply the work performed from the operator and the system. A clear view of the tasks that the operator has to perform, it is useful to figure out the system user expectation. We used the user viewpoint to select the characteristics for the specific quality model. The quality requirements are based on the system aspects pointed out from the user. The project staff conducted this work by interviews, check lists and user reviews. The following sections show and explain the quality characteristics identified for each system component.

*Table 1. Quality requirements for the train control system.*

Product	Component	Quality Characteristic	Quality Sub characteristic	Requirement Level
Traffic Control System	System Monitor	Reliability		Very High
		Functionality	Function completeness	High
			Function correctness	High
		Efficiency		Medium
		Maintainability	Corrective Maintainability	High
	Adaptive Maintainability		Very High	
	Decision Support System	Reliability		Medium
		Functionality	Function completeness	High
			Function correctness	High
		Efficiency		High
		Maintainability	Corrective Maintainability	Low
	Adaptive Maintainability		Very High	
	Man Machine Interface	Reliability		Very High
		Usability	Operability	Very High
			Learnability	High
Maintainability		Corrective Maintainability	High	
	Adaptive Maintainability	Very High		

### 3.1. External characteristics

The external characteristics identified for the final system are Functionality, Reliability, Usability, Efficiency, and Maintainability, they are the applicable subset of the external quality characteristics suggested in the standard ISO 9126 [11].

The external reliability will be evaluated on the basis of the product behaviour during the test performed in the user environment (acceptance test). The failures occurred during execution will be recorded specifying severity levels, time behaviours, affected components, and operational profiles. There are different reliability requirements for the three system components (System Monitor, Decision Support System, Man Machine Interface). For each failure, the affected component will be identified to evaluate that the target reliability of each component has been reached.

The external usability focus on learnability (the capacity of the software product to enable the user to learn its application) and operability (the capacity of the software product to enable the user to operate and control it). Learnability will be evaluated by measuring the

effort needed to learn a set of operator task. In particular the evaluation will concern the training time required to perform a specific task in a predefined percentage of the time required to perform the same task before training. Operability will be evaluated by measuring the user failure frequency due to the Man Machine Interface. This Measure evaluates indirectly if the Man Machine interface is implemented correctly. Moreover, The aim is to asses if the Man Machine Interface is misunderstanding or misleading for the operator.

The efficiency is evaluated by measuring the response time for each system component. From the viewpoint of efficiency, the most important component is the Decision Support System. It has to perform all the system functions implemented in order to predict system behaviours and to suggest solutions to the operator, as well. These functions could be very expensive in terms of time. Therefore, it is needed to fix a maximum time in which the implemented functions should turn out results.

The completeness and correctness of the system functions are evaluated by the attributes into the functionality characteristic. The need of new functions is measured to evaluate the completeness of the system functions. In stead, the correctness is evaluated by measuring the need of modifications of the functions already implemented.

The maintainability is considered as an external characteristic, because the customer (FS) asked for a system easy to maintain during execution. In particular, it is important that the system has to be easy to recover from failures. Moreover, it has to be easy to change, because the user could have new needs that depend on the incoming situations. Maintainability consists of two sub characteristics corrective and adaptive maintainability. The first refers to the maintainability of the system in case of failures. The corrective maintainability is evaluated by measuring the recovery time needed for failure. We distinguish two different kind of failures, software failures and operator failures due to Man Machine Interface. The faults related to the operator failures due to Man Machine Interface are those that affect the understandability and leadability of the Man Machine Interface. The corrective maintainability refers to the maintainability in case of requests of changes. It is very important for the Man Machine Interface and the System Monitor. In the user-centred design approach, you expect several modifications in the system (displays and functions) because of the continuos evaluation and refinement of the prototypes with the user. Thus, the aim is to asses whether it is convenient to modify or rewrite modules. The modification productivity is based on the comparison between the man power needed to implement an amount of code (e.g., expressed in Line Of Code) and the man power needed to modify the same amount of code.

### **3.2. Internal characteristics**

The main aim to collect internal measures is to monitor and control the progress of the product quality. It is assumed that controlling internal quality attributes is important for achieving external quality requirements, even if there is not any proved relation between internal attributes values and external quality attributes values. The practice has defined this heuristic. For example, the level of reliability obtained by each system component is estimated using reliability growth models during system testing [4, 5, 6]. The information obtained can be used to drive testing influencing controlling in this way the software development process.

We do not describe the internal characteristics of the system for reason of conciseness. These characteristics are shown together with their target values in table 2 for the system component System Monitor.

Table 2. Internal quality characteristics and requirements for the System Monitor.

System Component	Quality Characteristic	Quality Subcharacteristic	Quality Subsubcharacteristic	Attribute	Target
System Monitor	Reliability	Estimated Reliability		Mean time between failures	$\geq 1*10E3$ hours
		Correctness		Module fault density	$\leq 10$ faults per module
		Test Accuracy	Test Coverage	Function coverage	$= 100\%$
				Branch coverage	$\geq 70\%$
		System Evolution Trend		Trend of module modification	to be defined on first changes
	Trend of requirement modification			to be defined on first changes	
	Functionality	Function Evolution Trend		Trend of requirement modification	to be defined on first changes
	Maintainability	Corrective Maintainability		Number of modules changed per fault	$\leq 2$ modules
		Modularity	Coupling	Number of modules calling a module	$\leq 10$ modules
				Number of modules called by a module	$\leq 30$ modules
				Simplicity	Module size
		Number of modules	$\leq 100$ modules		
		Analysability	Code Readability	Static graph theoretic complexity	$\leq 50$
Density of comments				$\geq 1/10$ (Line of comments/LOC)	

The quality characteristic named System Evolution Trend has been identified because of the specific development process used in this project. We discussed that the user centred design approach is an iterative process with frequent evaluations of the work in progress with the user. For this reason, both the system requirements and its implementation (in modules) will be frequently updated. This frequent requirement up-date is usually considered as a negative development process behaviour affecting the final product reliability. For example, the Software Maturity Index [9, 10] is used to quantify the readiness of a software product. Changes from a previous release to the current one are an indication of the current product stability. In general, changes will produce a low level of the Software Maturity Index, this means that the software product is not stable. This is not the case in a user centred design approach where an evolution of the system requirements is part of the approach. By controlling the System Evolution Trend we want to check how the system requirements evolve with time and if this evolution follow a logical, converging path. The final hypothesis is that the System Evolution Trend function shape and the Estimated Reliability function shape are not mutually dependent.

#### 4. Selection criteria and critical issues

In order to collect the information needed for the quality model, we analysed the metrics proposed by several standards and guidelines (ISO/IEC 9126, IEEE std 982.1, NISTIR

5459). A selection of metrics is associated to the attributes of the quality model. This section shows the criteria used for the selection of the metrics. Some criterion is derived from the problems encountered during the project.

A basic set of criteria points out that metrics have to be objective, observable, repeatable and predictive. This implies that any metrics have to be based on physical properties of the product. In order to be objective and observable, they should measure features of the objects that form the product. A well-defined metric has to be repeatable. This means that if a measure is taken several times on the same product, every time you should obtain the same value. Moreover, the use of software metrics aims to predict the quality reachable for the software characteristics. Therefore, you should be able to predict the final quality achieved through software metrics.

Another selection criterion is the ability to collect the required data for the metric. The simplicity of data collection is one reason Line Of Code remains a popular metric despite its deficiency. The data collection activity of many other metrics can be automated, and therefore, may be less costly to implement. The availability of tools in order to automate the data collection activity is the main reason to select some metrics in stead of others.

One important aspect is that metrics should provide data valid across further projects. That is, the data collected on different projects should be comparable. Otherwise, it is not possible to increase experience. Past experiences are very important especially during first phases of projects. The most effort is needed to define a quality model and the quality requirements for a new project. The idea is to obtain feasible targets for a new project through previous reached values in similar projects (i.e., projects based on the same quality model).

Some metrics can be used to monitor the product progress as well as the design process. This aspect is critical in the project reported. In fact, the main problem is to define suitable metrics for the user centred design approach. The analysed metrics are misunderstanding and misleading for this approach. In order to monitor the design process, metrics should point out the basic properties of the process used. They have to turn out meaningful values in order to analyse behaviours related to the design process (e.g., System Evolution Trend and Function Evolution Trend). The user centred design approach takes into account human factors. Most of the metrics are not able to measure human factors concerning the software product. We point out the need for new metrics related to the design process. In the case of the user centred design approach, task analysis can be useful to define and validate the metrics proposed. In general, it is needed to define new metrics related to the design process. Because, the design process affects the progress of the system. Therefore, the quality of the product could be affected from problems related to the design process.

## **5. Data collection activity**

We introduced the organisation of a quality plan for the project. The last part of this plan is the data collection activity that will start soon. The data collection activity in the project uses tools. The most important tool used in the project is the SQUID tool [1, 2] that provides support for the SQUID approach to software quality. SQUID (Software Quality In Development) is a quantitative approach to software quality control and evaluation. The SQUID tool allows you to define your software quality model (which can be based on an existing standard such as ISO 9126) and to model your own software development process. The development process consists of a set of development objects. SQUID has three different type of development object, review points, development activities, and deliverables. You can then associate attributes with development objects. When you start planning a new project, you need to refine your quality requirements to the level at which you can set

quantifiable targets for each relevant external attribute. Then, you can set targets on the internal attributes you want to monitor during development. You can store actual measured values in the SQUID database throughout product development and analyse the collected data to determine how your project is progressing with respect to its quality targets. SQUID supports quality prediction by using statistical technique to predict final product quality in terms of the operational behaviour of the software when it is used. The predictions are used to support initial feasibility studies and the final actual measures are used to support quality evaluation of the final software product. SQUID supports quality control by helping you to set targets for internal attributes during development, and to monitor progress against those targets. In addition, attributes associated with product items (e.g., modules) are used to identify unusual items that may be a risk to product quality.

Other tools used are those to automate the data collection activity. Usually they are static analyser of the code. Tools that analyse dynamically the software are used during testing in order to collect data relative to the test coverage. When many tools are used in a project, a critical issue is the integration of tools. In particular if it is needed to transfer/acquire data among them, they should support input/output of ASCII files.

**Test Report - Form 1**

Date: \_\_\_\_\_ N°: \_\_\_-\_\_\_  
 Author(s): \_\_\_\_\_  
 Software Tested: \_\_\_\_\_

Test Sections

TS	Start Time	Execution Time	# Failure(s)	# Test Case

Failures

Failure	TS	Start Time	Elapsed Time	Type	Severity Level	Components	Description

*Figure 3. One of the forms used for the collection activity during testing.*

The report organisation/planning is one of the most important phases of the data collection activity [13, 14], because most data are collected through reports. The design process influences the planning as well as the report contents. In the case of the user centred design approach, task analysis results are pointed out into reports together with the design aspects. The fact that the process is iterative has to be taken into account for the report organisation. In fact, it can be difficult to identify the reports relative to some implementation, and vice versa. For example, a release of the prototype within the project is identified by a software requirement report and a set of scenarios produced through task analysis. Thus, there could be different prototype solutions based on the same software requirements, but on different sets of scenarios. This example shows the importance of the organisation/planning of the reports for the consistence and coherence of data. Figure 3 shows an example of report used for the collection of data during testing.

## 6. Conclusion

The paper shows a practical experience to define a quality plane for a software project dealing with the User-Centred Design approach. The definition of a quality model for a specific project is still quite difficult. Moreover, there is not guidance for the interpretation of

metrics within a specific design process. There exist specific problems related to the iterative design process (i.e., user centred design approach). An iterative process implies an evolution of the system requirements. In particular, several modifications are expected, and they should be taken into account carefully by software metrics. Some metrics can be misunderstanding or misleading into this approach. Metrics should be updated with user centred aspects. In fact, there are aspects relative to the interaction between user and machine. These aspects should be measurable in order to obtain a product wide accepted from the user. Task analysis can be useful to evaluate and validate new metrics. New metrics have to be validate trough further (similar) projects. Further works should investigate metrics in specific design context.

## References

- [1] Kitchenham, B., Linkman, S., Pasquini, A., Nanni, V., "The SQUID approach to defining a quality model", *Software Quality Journal* 6, 1997, pp. 211-233.
- [2] Kitchenham, B., Pasquini, A., Anders, U., Bøegh, J., De Panfilis, S., Linkman, S., "Automating software quality modelling, measurement and assessment", *ENCRESS '97*.
- [3] Kitchenham, B., Pfleeger, S.L., Fenton, N., "Towards a Framework for Software Measurement Validation", *IEEE Transaction on Software Engineering*, VOL. 21, NO. 12, December 1995, pp. 929-944.
- [4] Musa, J.D., Iannino, A., Okumoto, K., "Software Reliability: Measurement, Prediction, Application", McGraw-Hill, 1987.
- [5] Pasquini, A., Crespo, A.N., Matrella, P., "Sensitivity of Reliability-Growth Models to Operational Profile Errors vs Testing Accuracy", *IEEE Transaction on Reliability*, VOL. 45, NO. 4, 1996 December, pp. 531-540.
- [6] Del Frate, F., Garg, P., Mathur, A.P., Pasquini, A., "On the correlation between code coverage and software reliability", *International Symposium on Software Reliability Engineering (ISSRE '95)*, pp. 124-132.
- [7] Basili, V.R., Rombach, H.D., "The TAME Project: Towards Improvement-Oriented Software Environments", *IEEE Transaction on Software Engineering*, VOL. 14, NO. 6, June 1988, pp. 758-773.
- [8] Rai, A., Song, H., Troutt, M., "Software Quality Assurance: An Analytical Survey and Research Prioritization", *J. Systems Software*, 40, 1998, pp. 67-83.
- [9] IEEE Std 982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software".
- [10] IEEE Std 982.2-1988, "IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software".
- [11] ISO/IEC 9126, "Information Technology - Software quality characteristics and metrics".
- [12] Salamon, W.J., Wallace, D.R., "Quality Characteristics and Metrics for Reusable Software", *NISTIR 5459*.
- [13] ANSI/IEEE Std 829-1983, "An American National Standard - IEEE Standard for Software Test Documentation".
- [14] Wallace, D.R., Peng, W.W., Ippolito, L.M., "Software Quality Assurance: Documentation and Reviews", *NISTIR 4909*.