Squid is a method and a tool for quality assurance and control that allows a software development organization to plan and control product quality during development. The Telescience software development project used it to build a remote monitoring and control system based in Antarctica.

A Method for Software Quality Planning, Control, and Evaluation

Jørgen Bøegh, Danish Electronics, Light, and Acoustics
Stefano Depanfilis, Engineering Ingegneria Informatica
Barbara Kitchenham, Keele University
Alberto Pasquini, ENEA

An increasing number of software process and product standards emphasize the need for measurement. ISO 9001, for example, provides guidance for monitoring and controlling product and process characteristics during both production and installation.[1] However, standards provide little guidance as to what exactly users should measure and how to use the results to support the development of high-quality software.

Furthermore, measurement cannot be defined independent of context. A metric set judged valid on one project may lead to poor quality or high development costs when applied to another project.[2] When quality is measured, several factors come into play, including product characteristics (such as size), process maturity level of the company developing the software product, its development environment (such as the design methodology and CASE tools used), and the development team's skill and experience.[3]

0740-7459/99/$10.00 © 1999 IEEE

The Telescience Application

The Telescience Project was launched in 1993 as part of the Italian National Antarctic Research Program. Its aim is to develop an advanced supervision and telecontrol system to remotely perform experimental activities in Antarctica during the austral winter. The Squid method and toolset have been used to develop the software used in two of the Telescience systems: the control software of the Electric Power Supply and Distribution System, and the software of the Remote Man–Machine Interface.

The Electric Power Supply and Distribution System

The Electric Power Supply and Distribution System is located at the Italian base in Antarctica. This subsystem ensures that electricity produced by the on-site diesel generators is available to the equipment, scientific instruments, and systems running during the winter in Antarctica. The experimental activities there are intended to run unmanned for eight months, and this system must guarantee the power supply without interruptions. The system has severe requirements for reliability, availability, maintainability, and fault tolerance capacity. The real-time control software is duplicated. If the software fails catastrophically in both versions of the power supply subsystem, a back-up battery power system can run for one hour to provide the required energy.

The Remote Man–Machine Interface System

The main goal of the Remote Man–Machine Interface system (located in Italy) is to guarantee that scientists and operators can perform scientific activities and remotely control the base functioning during the austral winter. This system will permit scientists to remotely program, schedule, and control experiments. The system will also support the remote control of the station and its instruments, and will simulate environmental and operating conditions for training purposes.

Defining measurement in the specific context of an organization means using defined data collection rules and metric sets. Comparison of measurement values should be limited to projects with similar characteristics. This suggests the need for a method to control, assess, and predict product and process quality, which can be tailored to an individual organization's needs and based on its own databases.

The Software Quality in Development project, supported by the European Commission's Esprit program, has developed Squid to fill this need. Squid is a method for quality assurance and control, and includes a toolset to support application of the method. Using Squid, a software development organization can use measurement to

- plan and control product qualities during development;
- learn in a deeper and more systematic way about the software process and product, and feed the appropriate experience back into current and future projects; and
- evaluate final product qualities.

We have applied Squid to a software development project called Telescience (see the boxed text, "The Telescience Application"). This project aims to automate a scientific station in Antarctica, to monitor and control scientific experiments remotely during the Antarctic winter.

The Squid Approach to Quality

Squid defines *quality* as the operational behavior of a product required by its users. This is consistent with the standard approach to quality modeling taken by ISO 9126,[4] which defines quality as a set of product characteristics. Characteristics that govern how the product works in its environment are called *external* quality characteristics, which include, for example, usability and reliability. Characteristics related to how the product was developed are called *internal* quality characteristics, and include, for example, structural complexity, size, test coverage, and fault rates. Examples and descriptions of these characteristics can be found elsewhere.[4,5]

The Squid approach proposes defining product quality requirements by establishing targets for the external quality characteristics. This approach then proposes pursuing those targets during development by defining and monitoring targets for internal quality characteristics. This can be done using conventional software measures of size, fault rates, change rates, structure, test coverage, and so on, taken early in product development.

Squid helps in establishing relationships between internal and external quality characteristics, using experience from similar past software devel-

opment projects. This past experience is also used to assess the feasibility of quality requirements before starting the project, and to identify and set targets for internal quality characteristics. Past experience must be stored in a consistent way to allow comparability across different projects. The Squid Data Model and Quality Process provide a rigorous and organized method for identifying, defining, collating, and comparing quality measures.

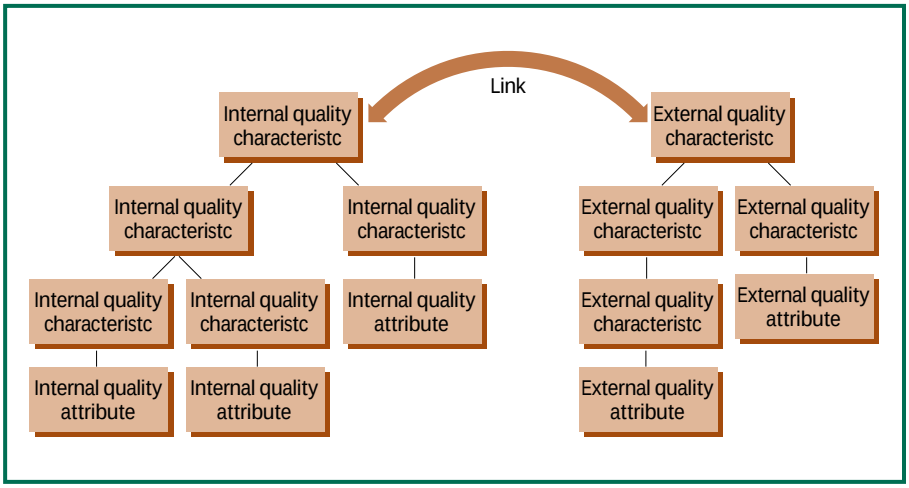


Figure 1. Possible refinement of the quality characteristics Internal and External Maintainability in the Squid quality model.

The Squid Data Model

Development processes vary from company to company, and even a single company may commonly use several different development models. These differences limit the possibility of using metric sets across different projects. For example, it is not meaningful to compare the number of lines of code developed using structured versus object-oriented techniques. Process differences also affect the relation between measures and quality characteristics. For example, the influence of "size" (measured as LOC) on software maintainability differs for software developed in an OO development process versus a Waterfall one.

An organization using Squid specifies the project objects (deliverables, milestones, modules, and so on) produced by the development process it uses. Project objects are the elements on which measurements will be taken. Squid requires mapping quality requirements onto quality characteristics and hence onto quality attributes (that is, measurable properties) in the framework of a quality model like that proposed in ISO 9126. Quality requirements then serve as a set of targets for the external quality characteristics. Quality models also identify internal quality characteristics that influence the external ones. The organization then monitors and controls internal quality characteristics through project measures to achieve quality requirements.

Representing the development process

Representing the development process in the data model characterizes the measures to be taken. The Squid data model identifies the essential characteristics of the development process such as the type of development model and the language adopted. Squid also models project objects, classified as deliverables, development activities, or review points:

- *deliverables* (such as specifications and code) are produced during the life of a project,
- *activities* produce deliverables, and
- *review points* are control intervals used in a particular development model.

Quality measures are taken of the deliverables at the specified review points.

Representing the quality model

Squid specifies a quality model in terms of quality characteristics, which are refined until they are directly measurable; they are then referred to as *quality attributes*. Quality characteristics and attributes can be internal or external; the user identifies how internal characteristics influence external ones by linking them in the data model. Figure 1 shows two related quality characteristics and how they are refined in the quality model.

Except for extremely simple projects, different parts of a software product usually have very different behavioral requirements. For example, the interface component has high usability requirements, whereas the database component has high integrity requirements. Squid distinguishes such different parts using the concept of a *product portion*. A single product comprises a set of product portions; different product portions have different quality requirements. This can help reduce conflict among

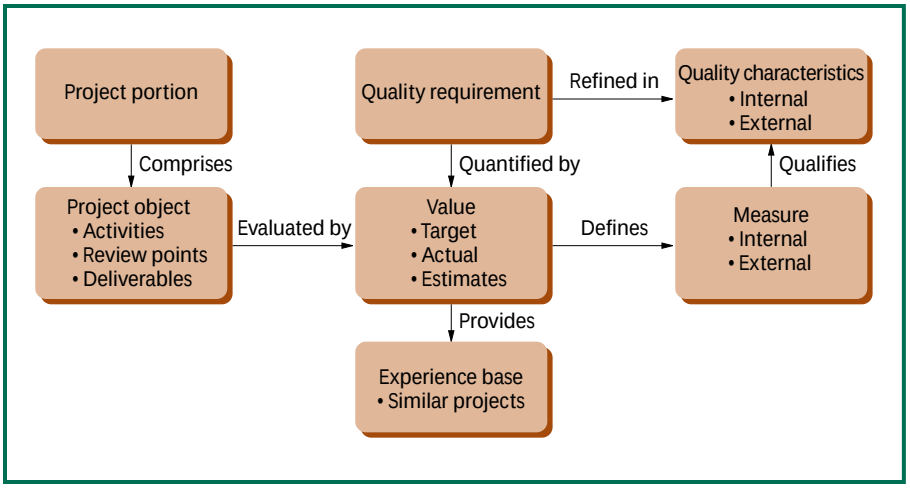


Figure 2. Simplified representation of the Squid data model.

differing quality requirements. Whereas it may be impossible to deliver extremely high operational requirements in many different and sometimes conflicting dimensions for a product as a whole, it may be feasible to deliver stringent quality requirements for the individual product portions that genuinely require them.

Representing measures

The measures collected through Squid can be actuals, estimates, or targets. An *actual* is the current value measured for a specific quality attribute. An *estimate* can be obtained from past data or through the Squid toolset's estimation facilities. *Target* values are specified during the quality specification phase. Measures can be expressed in different units; for example, lines of code is a product code length unit, elapsed days is a project duration unit, and working years is a personnel experience unit.

The Squid user defines how to take measurements in *counting rules*, which clarify and regulate all the conditions that could affect the measures' objectivity. For example, the measure of program length can be collected as lines of code, but a counting rule must confirm at what development stage the measure is to be taken. Counting rules ensure that measures are repeatable and are comparable with those of the historical database.

Integrating components of the Squid data model

All internal and external attributes belonging to a specified quality model must be allocated to the project object types belonging to a specified development process and associated to units and counting rules (project measures) in the Squid data model. Thus, the attribute "structural complexity" in the quality model must be associated with an appropriate project object type in the development process model—for example, a module in a Waterfall development model. Figure 2 shows the resulting organization of the Squid data model.

The Squid Quality Process

The Squid quality process includes quality specification, planning, control, and evaluation. Squid provides user interaction and support in several ways.

Quality specification

Quality specification means establishing the software product's quality requirements. There are three main activities involved:

♦ *Select the quality model*. Specifying the quality requirements takes place within the framework of a specific quality model. An organization using the Squid approach needs to define the quality models used to develop its software products—for example, an international standard like ISO 9126 or a company-specific model. When starting a new project, the organization must determine which of the available quality models and development processes should apply to the current project.

♦ *Identify product portions, quality requirements, and targets*. The product being developed must be defined in terms of a set of independent product portions, that is, parts of a software product with different quality requirements. Each operational requirement is then associated with the product portion to which it applies. The Squid toolset supports the association of each requirement with a quality characteristic in accordance with the selected quality model. Once a match is made between a quality characteristic and a requirement, the toolset identifies one or more external attributes, derived from the quality model, on which targets can be set.

♦ *Perform feasibility analysis*. Squid helps users evaluate the feasibility of the external attribute targets, based on comparing achievement of similar

Relationship with Other Methodologies

Squid is not the only quality methodology, nor is it unique in suggesting the use of measures to assist software development.

Process Improvement Methodologies

Process improvement methodologies such as CMM,[1] ISO 9001, and SPICE[2] identify the features essential to good software development. They do not define a specific development process but instead guide software producers in defining their methods and implementing appropriate procedures to manage software development. Their goal is to improve the development process—they do not claim to improve individual products except as a by-product of better processes.

These methodologies advocate the use of measurement to improve project and quality management. Squid complements them by providing a framework within which quality measures can be stored, analyzed, and used constructively to assist software development.

Metrics Methodologies

Of the several metrics methodologies in current use, one of the most widely known and applicable to quality is the Goal-Question-Metric paradigm.[3] GQM provides a means of deriving a measurement program from a statement of the business goals to be achieved. Once the program goals are stated, they are used to generate questions that the measurement program must address. The questions enable the user to identify the metrics needed to answer the questions. GQM is a high-level framework that does not address specific software engineering issues. Within this generic framework software developers must specify and document their measurement program.

Product Quality Methodologies

Product quality models include ISO 9126 and Euromethod.[4] ISO 9126 defines a generic quality model and various supporting guidelines. Squid supports using ISO 9126 exactly as specified in the standard. However, Squid's flexible quality modeling facilities mean that a user can adapt the ISO quality model or use any other model appropriate for the type of applications the organization produces. ISO 9126 and other quality models recommend quality specification and evaluation actions but do not specify how they should be performed. Squid provides a procedure and a toolset for undertaking the quality actions recommended in the standard.

In addition, Squid's rigorous approach to measurement ensures full definition of measures. The user specifies the factors (counting rules, project objects, and product portions) needed to ensure that data is reliable and comparable. This forms the basis for feasibility analysis, quality specification, and quality planning. In addition, Squid extends the ISO 9126 philosophy to quality planning and control.

Euromethod approaches software quality through identification and control of product deliverables and intermediate deliverables. Euromethod is not itself a measurement-oriented approach. Thus, we see Squid and Euromethod as supporting technologies since Euromethod defines the product deliverables and Squid models them in order to associate measures with them.

Squid has some similarities with other quality and measurement-based technologies, but its specific product quality focus differentiates it from other methodologies. Squid compels its users to define their development process, quality models, and measures. It is, therefore, supported by process initiatives such as CMM and ISO 9001 that are forcing organizations to adopt better-defined development standards and procedures.

References

1. M.C. Paulk et al., "Capability Maturity Model, Version 1.1," *IEEE Software*, July 1993, pp. 18-27.
2. T.P. Rout, "SPICE: A Framework for Software Process Assessment," *Software Process—Improvement and Practice*, pilot issue, 1995, pp. 57-66.
3. V.R. Basili and H.D. Rombach,"The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. Software Eng.*, Vol. SE-14, No. 6, June 1988, pp. 758-773.
4. M. Gibbons, C. Mackie, and M. Pfeiffer, "Opportunities Deriving from the Combined Use of Euromethod and SPICE," *Software Process Improvement and Practice*, Vol. 1, No. 2, 1995, pp. 115-135.

past projects with the targets of the current one. The toolset analyzes the historical database looking for similar past projects; it judges the level of similarity by examining the development process used for past projects and their external attribute targets. The user can participate by introducing search constraints and removing certain projects from consideration. Squid then analyzes the actual values of the external attributes achieved by the selected past projects and compares these with the new product's target values. If the actual values differ from the required values, the normal development process may not be able to deliver a product that meets the stated quality requirements. Of course, this feature requires a well-populated historical database.

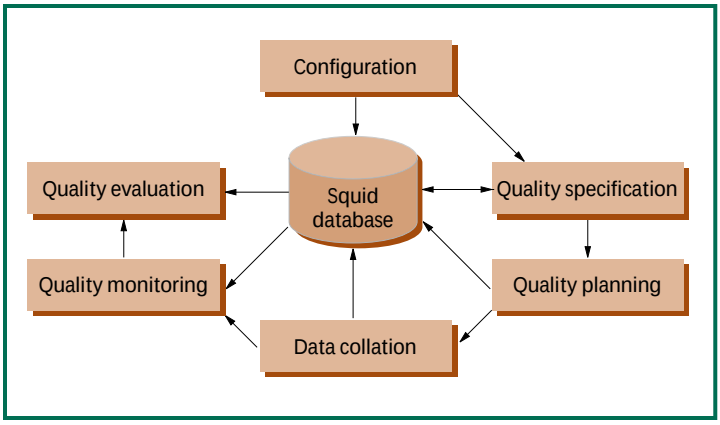


Figure 3. Quality process activities and their relation to the Squid database.

Quality planning

Quality planning involves deciding on a suitable development process and setting target values for internal attributes.

♦ *Select the development process for each product portion.* Since product portions have different quality requirements, the development team may need to adopt different processes to implement them.

♦ *Assign target values to internal quality attributes.* Quantitative targets are set for the internal quality attributes identified in the quality model. Attributes are then associated with intermediate activities and deliverables (project objects), against which progress will be monitored and controlled. The Squid toolset helps users assign targets to internal measures by allowing them to view the software quality attribute targets achieved on similar past projects.

Quality control

Squid supports monitoring progress throughout development using internal software measures associated with deliverables and activities related to each major review point in development.[6] Quality control activities include the following:

♦ *Identify deviations by comparing target values with actual values of the internal quality attributes.* At each project review point, when certain quality attributes have been measured, the current quality status of the project can be assessed. This involves comparing each internal quality attribute with its planned target and identifying any significant deviations.

♦ *Identify software components with unusual attribute values.* These components should be analyzed more carefully to ensure early identification of problem areas. The Squid toolset automates the detection of anomalous components to identify project objects with unusual combinations of internal attribute values.

Quality evaluation

In Squid, quality evaluation assures that a product satisfies its specified requirements. The development team can also analyze the quality results of a project to provide feedback for the development process. Squid provides a basic evaluation mechanism based on

♦ measuring the actual values of the external attributes for each product portion,

♦ comparing each actual value with its target value, and

♦ reporting quality shortfalls.

Figure 3 shows the sequence of the Squid quality process activities and their main relation to the Squid database.

Application of Squid to Telescience Software

The Squid approach is being used by the Antarctica Telescience project for development of the Electric Power Supply and Distribution System and the Remote Man-Machine Interface. The quality specification and quality planning activities of these two systems have been successfully completed, and quality control is in progress. We report here the experiences and problems encountered during these activities.

Quality specification

Using the Squid approach, establishing the quality requirements for the Telescience project involved three steps.

Select the quality model

The project team started by reviewing the ISO 9126 quality model. This standard specifies six quality characteristics, but it is unrealistic to assume that a single set of quality characteristics is necessary and sufficient to describe the quality requirements for every project. In the case of the Telescience project, the ISO characteristics were not exactly those that best described the systems.

The project team therefore defined a specific quality model, modifying and integrating that proposed

in ISO 9126, that identified the quality characteristics and the associated external attributes. Table 1 shows the quality requirements and targets for one product portion, the Electric Power Supply subsystem. The first three columns show the set of external characteristics and external attributes associated with the quality characteristic Maintainability. Each element of the quality model and the counting rules to be used for each attribute were defined in the database.

Table 2 shows the targets for each internal attribute of Maintainability for the Electric Power Supply subsystem.

We found the Squid method to be very sensitive to the quality model identified, and especially to the characteristics of the external and internal attributes selected. Because Squid is based on learning from the experience of similar past projects, attributes must be as robust as possible against variation of the project characteristics; that is, they must remain significant even across different projects. A project-specific attribute will not have significant value for other projects, even if they are only slightly different. The quality characteristic "product size," for example, can be expressed in terms of attributes such as code size or function points. The first is sensitive to changes in the implementation language while the second is not, which means the second attribute is significant even across projects that use different languages.

Identifying quality attributes with such characteristics was not easy for the Telescience project, which is a real-time control system used under unusual operating conditions. This system has peculiar requirements for reliability, maintainability, and interacting with the environment; some of these were mapped into application-specific external attributes. For example, the whole Antarctica station relies on the energy produced under the control of the Electric Power

Table 1
Quality Requirements and Targets for Electric Power Supply Subsystem

External Quality Characteristics		External Attributes	Unit	Upper Target
Maintainability	Corrective maintainability	Recovery time	Average number of hours required for system recovery (*)	1
			Average number of hours required for software recovery (**)	48
		Fault containment efficiency	Average number of modules modified per error fix	1.2
	Adaptive maintainability	Modification productivity	Productivity ratio	1.5

* System recovery refers to failures affecting the system's ability to produce power.
** Software recovery refers to failures affecting the software's ability to control the power production.

Table 2
Targets for Internal Attributes for Electric Power Supply Subsystem

Internal Quality Characteristics			Internal Attributes	Lower Target Value	Upper Target Value
Maintainability	Modularity	Cohesion	Average number of modules changed per fault corrected during integration testing	–	1.2
		Coupling	Modules calling a module (fan-in)	–	10
			Modules called by a module (fan-out)	–	30
			Average common data access	–	5
		Simplicity	Average module size	–	200
			Number of modules	–	100
			Cyclomatic complexity	–	60
	Analyzability	Document readability and completeness	Conformance to documentation standards	–	8
			Document readability	–	17
		Code readability	Density of comments	0.1	–

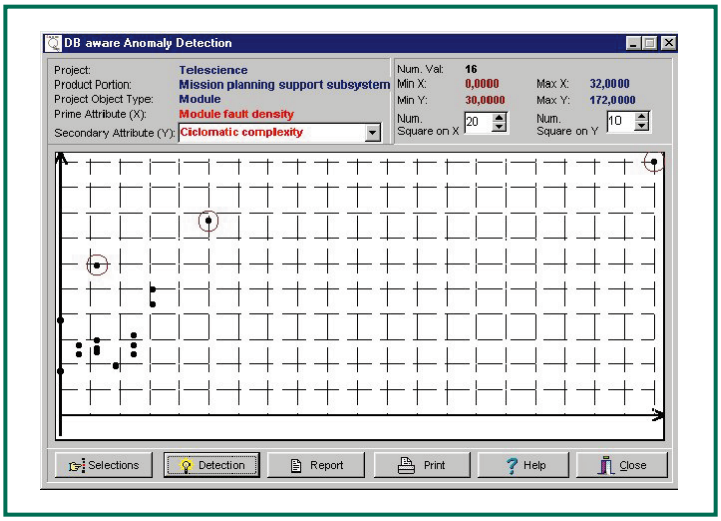


Figure 4. Example of a scatterplot used to identify anomalous modules.

Supply and Distribution System. If this system has a catastrophic failure, a back-up battery can power the station for about one hour, during which time the failure must be detected, the fault removed, and the power production restored. These severe, very specific requirements demanded a powerful way to define their external attributes and to evaluate them.[7]

Identify product portions, quality requirements, and targets

The Telescience project had two sets of quality requirements for the Electric Power Supply and Distribution System because different subsystems had different reusability requirements. Hence the team defined two product portions: the Electric Power Supply subsystem and the Power Distribution subsystem. The design and some of the code of the power supply subsystem will be used in a commercial control board for the power supply hardware, whereas the other subsystem is not expected to be reused.

Perform feasibility analysis

After quality requirements are specified, the Squid toolset supports feasibility analysis by comparing the quality achievements of similar past product portions with the quality requirements of the current product portion. Because the Telescience project is the first to use the Squid approach, however, no past experience was available.

Despite this significant limitation, the Squid approach supported quality specification by forcing software engineers to analyze the system quality requirements in-depth, solve conflicts between functional and nonfunctional requirements, and identify quality requirements for each portion of the system.

Quality planning

Squid defined two main activities for quality planning.

Select the development process for each product portion

Based on the application's characteristics and the organization's past experience, the Telescience team used only a conventional Waterfall model for all the product portions. This model was represented as a series of review point types together with a set of deliverable types and development activity types. The team introduced these into the Squid database and integrated them with the other components of the Squid data model, as described earlier.

Assign target values to internal quality attributes

The identification of the internal quality characteristics, attributes, and targets proved difficult in this project because of the lack of previous experience. Developers therefore relied on expert opinion, using as a starting point a set of attributes suggested by the Squid project staff. The ability of the Squid toolset to store and use past company experience should facilitate target-setting in the future.

Quality control

Squid outlined two main activities to help the Telescience team monitor the project's progress; both are still underway.

Identify deviations by comparing targets and actual values

The development team can identify deviations from plans by comparing target values with actual values of the internal quality attributes. They have already done so for internal quality attributes related to project objects delivered early in product development. While the Telescience project represented an experimental application of the Squid method, this feature significantly improved software engineers' ability to control product quality during development. This activity will continue throughout the Telescience design and development life cycle.

Identify software components with unusual attribute values

The Telescience team found this toolset facility particularly useful. Scatterplots relating two attributes for the same objects provide a useful visual aid for analyzing the project's large number of deliverables. Anomalous components can immediately be

detected and examined more closely to identify possible quality problems.

Figure 4 shows the scatterplot of the attribute "number of faults" (total number of faults detected in the module during testing) versus "cyclomatic complexity" for a subsystem's modules. Each project object (the modules in this figure) appears as a point within the scatterplot; users can click on the point to identify the project object and obtain more information about it.

Most software engineers believe that properties of the intermediate products of the development process affect final product behavior, but there is little scientific evidence of this link and no successful examples of using it to predict or control final product behavior. Squid aims to clarify this link by encouraging software developers to collect quality data and analyze the relation between properties of both the intermediate products and the development process with final product behavior. The prototype toolset developed and used in the Antarctica Telescience project is now being upgraded to an industrial prototype for more extensive trials. ❖

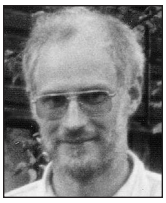
Acknowledgment

This was partially funded by the European Commission as the Squid (Software Quality in Development) and Valse (Validating Squid in Real Environments) projects. We thank the EC and all our colleagues who collaborated with us in those projects.

References

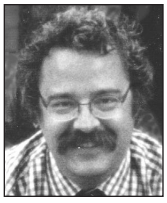
1. *ISO 9001 Quality Systems—Model for Quality Assurance in Design/Development, Production and Servicing*, Int'l Organization for Standardization, Geneva, 1994, pp. 652-663.
2. V.R. Basili, R.W. Selby, and T.Y. Phillips, "Metric Analysis and Data Validation across Fortran Projects," *IEEE Trans. Software Eng.*, Vol. SE-9, No. 6, Nov. 1983.
3. N.F. Schneidewind, "Minimizing Risk in Applying Metrics on Multiple Projects," *Proc. 3rd Int'l Symp. Software Reliability Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1992, pp. 173-182.
4. *ISO 9126 Information Technology—Software Product Evaluation—Quality Characteristics and Guidelines for their Use*, Int'l Organization for Standardization, Geneva, 1991.
5. N.E. Fenton, *Software Metrics: A Rigorous Approach*, Chapman-Hall, London, 1991.
6. B.A. Kitchenham, S.L. Pfleeger, and N.E. Fenton, "Towards a Framework for Software Measurement Validation," *IEEE Trans. Software Eng.*, Vol. 21, No. 12, Dec. 1995, pp. 929-944.
7. A. Pasquini, E. De Agostino, and G. Di Marco, "An Input-Domain-Based Method to Estimate the Reliability of Software," *IEEE Trans. Reliability*, Vol. 45, Mar. 1996, pp. 95-105.

About the Authors

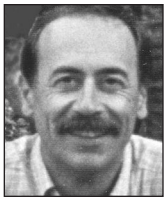


Jørgen Bøegh is a researcher in computer science at Danish Electronics, Light, and Acoustics (Delta) in Denmark. He received an MSc in mathematics and computer science from the University of Aarhus. From 1980 to 1985 he was with the Danish Defense Staff; since 1985 he has been with Delta (formerly Elektronik Centralen) in its Software Engineering Division. He has been involved in several European research projects in the areas of software quality and software certification. His research interests include software quality and software quality measurement. He is a member of ISO/IEC/JTC1/SC7.

Barbara Kitchenham is a principal researcher in software engineering at Keele University as well as a member of *IEEE Software*'s Editorial Board. Her main interest is software metrics and their support for project and quality management; she has written more than 40 articles on the topic as well as *Software Metrics: Measurement for Software Process Improvement*. She spent seven years at the UK National Computing Centre. Kitchenham received a PhD from Leeds University.



Stefano Depanfilis is the director of the Software Engineering Competence Centre at Engineering Ingegneria SpA, an Italian software house. His interests include software development process customization, support, monitoring, and control. He has been involved in several European research projects in the area of software measurement, software quality, and software process improvement; on some of these projects he served as project manager.



Alberto Pasquini is principal research investigator in software engineering at the Robotics and Information Technology Division of ENEA, the Italian national research agency for new technology, energy, and the environment. He received a doctoral degree in electronic engineering from the University La Sapienza of Rome. His research interests include software reliability, software quality, and reliability of the human computer interaction. He is a member of the program committee of several international conferences and will be program chair of the upcoming conference Safecomp 99 (Safety, Reliability, and Security of Computer Systems). He has published more than 40 papers on software reliability, testing, and safety.

Address questions about this article to Pasquini at ENEA, sp.088, Via Anguillarese 301, 00060 Rome, Italy; e-mail pasquini@cassaccia.enea.it.