

Software quality management and process improvement with the MODAL development methodology

J. Wright

*Method Support Department, Cegelec RED,
5 avenue Newton, 92140 Clamart, France*

ABSTRACT

The paper describes the adaptation of an existing software development methodology in order to introduce a number of software metrics intended to increase visibility of the development process and guide quality actions at the project and divisional levels. The chosen indicators are described, together with some initial experiences gathered from a pilot project responsible for validating the approach taken. A number of lessons learned are described, particularly relating to data collection and the need to carefully define elements of the programme in order to gain acceptance by the development team.

1. MODAL : A DEVELOPMENT METHODOLOGY FOR CONTROL ENGINEERING SOFTWARE

Cegelec is the electrical engineering branch of the Alcatel-Alsthom group, and realises around 40% of its turnover in the industrial process control and service sector including automation, regulation, instrumentation and supervision of processes. The systems and products developed use computers, programmable logic controllers (PLCs), communications networks and related electrotechnical equipment, and are driven by software operating in complex real-time embedded environments. In 1986, the Industrial Control Branch of Cegelec was faced with the realisation that software was playing a progressively more important role in the systems which it was developing, to such an extent that the principal business focus was moving from industrial equipment supplies to real-time software development. The Research and Development Division (RED) of the BPT branch responsible for developing new products and technologies was charged with investigating the adoption of a software development methodology for use on all industrial automation and control projects.

Having decided that buying in an existing methodology would require too much customisation and would fail to sufficiently build on the body of development expertise already acquired within the company, RED began development of a new methodology (to be named MODAL®) in conjunction

204 Software Quality Management

with an independent external consultancy group specialising in software engineering (OPL), and representatives from each of the application divisions of the Industrial Control Branch. Seven working groups, involving a total of 50 development engineers and managers, were involved in consolidating a draft version of the MODAL documentation which was made available in 1987.

The methodology includes a generic development process, a set of recommendations and procedures for each phase of the development life cycle, templates and checklists for work products to be produced throughout the project, and guidance on the use of particular methods and techniques for certain critical phases and activities. Over 3000 pages of documentation are organised into 4 levels, guided by a reference document which describes the standard life cycle and establishes a glossary of terms. Application of the first two levels (procedures and document templates) is mandatory upon all projects using MODAL, while the Quality Plan of a given project will identify which elements of the second two levels (recommendations and guidelines for choice and operation of "preferred" methods and tools) are to be employed upon a particular development.

Since becoming fully operational in 1989, the procedural parts of the methodology (development, management and quality procedures, overall life cycle process, document templates) are now used on all new control engineering developments within the Cegelec group, and some aspects (project and configuration management, quality assurance) have been applied retrospectively to several long-term projects which started before MODAL became operational.

The Cegelec group is structured as a set of application divisions, each of which specialises in a particular industrial sector, such as power production, power transmission, transport etc. Each application division using MODAL (currently all divisions within the Industrial Control Branch) has elected a Methodology Correspondent, who is responsible for monitoring the use of the methodology within his division and presenting findings and suggestions at regular meetings, as well as proposing representatives for working parties of the Methodology Improvement Group, which handles extensions and evolution of MODAL. These groups currently include representatives from divisions in France, Belgium and the UK.

The use of MODAL has been underpinned by a Method Support Department (DMS) within RED, including a Training School, which has delivered over 7000 man-days of training in the principles and practicalities of applying the MODAL methodology. A group of 10 consultants work with the application divisions to provide technical support and assistance to the projects using MODAL.

The subject of this paper is the adaptation of MODAL to support measurement of the software development process as an aid to quality management and process improvement. The following section describes the indicators which we chose to include in our metrics programme, based on our current quality improvement priorities and our previous experiences in software measurement, and outlines our plans for incorporating documentary and tool support into the MODAL package. Section 3 then outlines the project which was chosen to validate our approach, and section 4 discusses some problems which

we discovered and analyses the ways in which they could have been avoided, examining how we can build upon our experiences in generalising the procedures developed on the pilot project. Section 5 then resumes the results which we have achieved and discusses the future directions which our measurement programme may take.

2. A ROLE FOR SOFTWARE MEASUREMENT IN MODAL

The consistent application of MODAL within a division, supported by usage of the functions provided by the DMS support team (see §1), offers a sound basis for working towards attainment of a software development process consistent with the characteristics of level 3 of the SEI capability maturity model (Humphrey¹). The development process resulting from effective application of MODAL can be characterised by the following properties:

- a well-defined development life cycle
- project planning including estimation of software size and development effort
- systematic application of quality assurance and control activities
- a rigorous process for configuration and change management, including tracking of signalled anomalies.

As the principal challenges in passing to level 4 of the SEI model involve process measurement and analysis, and creation of quantitative quality plans, we sought to enhance MODAL so as to aid its users to gradually improve their development processes along the lines indicated by the structure of the SEI model, in providing support from within the methodology for measurement of development capability and setting of improvement targets.

Recommendations for project organisation under MODAL include the identification of a number of roles, amongst which we will particularly refer to the **project manager**, an independent **quality assurance manager** assigned to the project, and a **maintenance manager**, responsible for supporting the system or product developed after first delivery to the customer. As part of our quality improvement programme, we have identified the need to expand the Methodology Correspondent role within each application division to encompass process ownership, implying wider-ranging responsibilities for application of MODAL within the division. The **process owner** will actively monitor the level of quality achieved on developments and guide the tailoring of MODAL to best suit the particular requirements of the division (types of contract, types of software, level of dependability requirements etc). This is to be achieved through the consolidation and exploitation of software metrics collected upon the projects undertaken within his division.

Previous attempts to systematically introduce software measurements within a given application division had generally been focused upon use of a software complexity analyser such as Logiscope® to provide a certain number of code metrics, e.g. cyclomatic number (McCabe²). While these had been successfully used in certain contexts when specific measures and associated thresholds were mandated by the customer, for example upon development of the Engineering Management System for the Channel Tunnel (Pham Van³), no generally-applicable objectives were ever published and no real consolidation of results provided. We also investigated use of a quite complex hierarchical, heuristics driven model originally developed for use on on-board software for

206 Software Quality Management

weapons systems (Forse⁴), but rejected its adoption within the MODAL framework as being too heavy weight for systematic application. However, certain of the ideas embodied by the model (e.g. the McCall⁵ approach of structuring quality requirements into factors and criteria) were adopted in our Quality Plan templates, where critical factors and criteria must be identified, and mechanisms put in place to ensure that the project can respond accordingly to the requirements elicited.

In an attempt to gain wider acceptance this time, we preferred to move the focus of our efforts to the development process and to leave our set of measurements open to completion by complementary, more detailed measures such as code complexity, requirements stability etc. (see §2.3) We decided to concentrate on using graphical summary sheets to track a small number of simple indicators, developing analysis procedures for each summary sheet and for all interested parties. We settled upon the following indicators:

- (1) effort estimate accuracy (see § 2.1).
- (2) defect detection / correction effort, to study possible savings in the cost of detecting and correcting defects through their detection at the earliest possible stage in the development life cycle.
- (3) defect injection / detection rate (see § 2.2).
- (4) code review efficiency, to support trade off analyses of the relative efficiency of unit testing and formal peer review of source code.
- (5) component defect density (see § 2.3).
- (6) outstanding anomaly backlog (see § 2.4).
- (7) anomaly category, to evaluate the percentage of anomalies flagged by customers which are effectively due to software defects.
- (8) age of outstanding customer defects, to support achievement of objectives aimed at improved turn around times for fixing of defects detected after delivery.

2.1 Judging our ability to estimate

Project management with MODAL covers the four principal topics of estimation, organisation, planning and tracking. We wanted to provide better visibility on the effectiveness of our estimation process, which currently uses an estimation model based on COCOMO (Boehm⁶) with various cost factors being adapted to our development environment. Significantly, no mechanism has been put in place to refine the model through feeding back measures of its accuracy. By systematically comparing the effort estimated for each phase of the life cycle with that actually consumed during the project, we hope not only to allow the project manager to better track resource consumption, but also to allow the process owner to consolidate results across several projects and fine tune the estimation model to provide more realistic estimates. We plan to track the "effort estimate accuracy" indicator via the summary sheet shown in figure 1, which must be prepared and distributed by the project manager.

The figure shows values calculated for our pilot project (see §3). The "actual" values include final figures for the specification phase and current, intermediary figures for the other phases. Although all of the design documents have been completed (and coding of some sub-systems started) definitive effort figures for the design phase are not yet available, as design documentation of one of the sub-systems is still under review. The effort charged to date to the

integration phase involves mainly preparation of integration plans and integration of low-level libraries with some sub-system components.

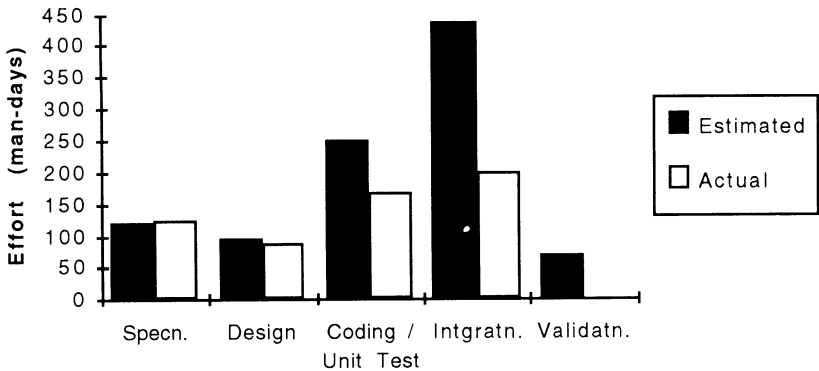


Figure 1 : Tracking Effort Estimate Accuracy.

2.2 Modelling the injection and detection of defects

We decided early on in our programme that the single most important indicator of the quality of our software was the number of defects contained therein. The quality of our development process could therefore be assimilated to rates of generation and removal of defects.

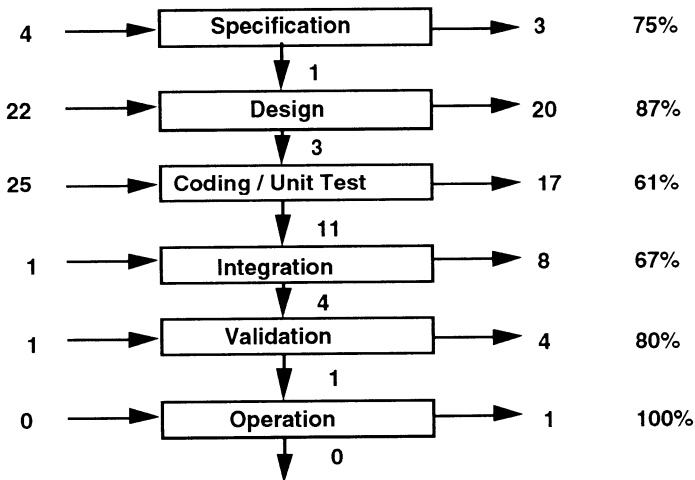


Figure 2 : Development process modelled as defect generation / detection rates.

Following the teachings of IBM (Jones⁷) we planned a two stage programme, based on the assumption that over a sufficiently large sample of projects, we could observe a constant defect density, this density depending directly on the quality of the development process. The first stage of process improvement consists in calculating the defect generation density and adapting

208 Software Quality Management

the defect **detection** process so as to remove an agreed percentage of these defects before delivery. A second stage is then to reduce the defect **generation** rate by adopting new technologies during the development activities (e.g. improved methods and tools). We therefore sought to model our development process in terms of rates of defect generation and detection.

After consolidating information from several projects, the process owner should be able to construct a model such as that shown in figure 2. Each box represents a phase in the development process, with the numbers shown on the left of the box representing the number of defects generated by the phase (per 1000 instructions delivered). The numbers on the right show similar rates for the number of defects detected during the phase, leaving residual defects (label on vertical arrows) which pass undetected into the following phase. Quantitative defect detection objectives can be developed in terms of the percentage of defects detected during any particular phase (numbers on extreme right).

Such a model can be instantiated by a quality manager at the start of a project, using the estimated size of the software to be delivered, in order to provide quantitative objectives for defect detection activities during each phase. In publishing the model, the process owner should associate statistical bounds around the values in order to accommodate variance between projects. This will allow the quality manager of a project to establish a defect detection plan such as the one shown in figure 3.

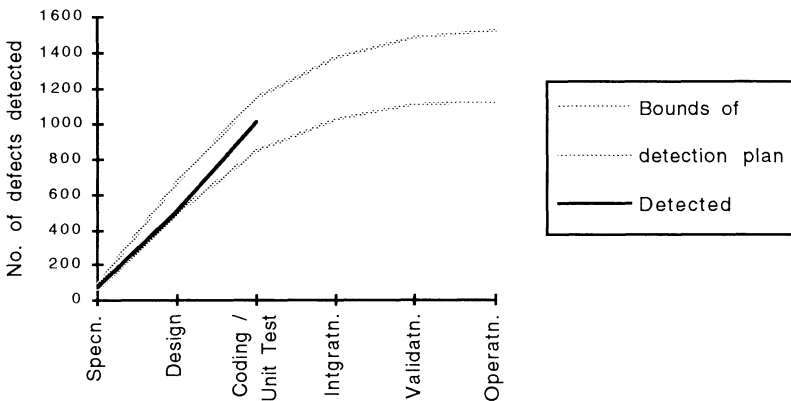


Figure 3 : Tracking achievement against a defect detection plan.

Tracking detection activities against the plan allows the quality manager to use objective criteria for exiting from a particular phase, and enables the maintenance manager to estimate the number of residual defects at the start of the maintenance phase. Clearly, we are still some way from being able to use this indicator to manage our projects, as the pilot project is merely gathering initial data to help in developing the model shown in figure 2. The data shown in figure 2 is therefore representative of figures published by companies undertaking comparable software developments (density of 53 defects per thousand shipped source instructions), while the data shown in figure 3

represents tracking of actual detection data against an instantiation of this model for a project of a similar size to our pilot project (see §3).

2.3 Identifying Error Prone Components

Our experience on previous projects has suggested that a significant proportion of defects are associated with a small number of components. We would like to be able to study more closely our coding activities, in order to identify the characteristics of such components exhibiting high defect densities. For those defects which originate in the software code, tracking their cause to individual components would help us to examine any trends which emerge and would aid in developing coding (or design) recommendations as well as focusing quality control activities on components which were anticipated as being problematical. A graph of defect densities by component such as that shown in figure 4 allows the project quality manager to identify candidate components for causal analysis. Such a component analysis must include examination of a number of additional measures which are not mandated in our list of indicators. These are left to the discretion of individual quality managers, as they will largely depend upon the particular characteristics of the given project.

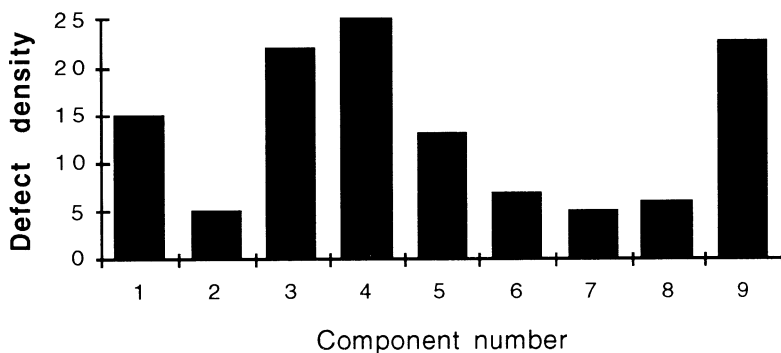


Figure 4 : Defect density by software component.

We have gained an initial experience upon another project where a series of "visibility programmes" were put into place in order to identify functional "hot spots" which were to be excluded from a cut-down first version of the system under development (a power dispatching plant). Having regrouped by function those components identified using a graph as in figure 4, we studied measures of size (number of source instructions), complexity (cyclomatic number and algorithm depth) and stability (number of modifications to source code) effected upon these components, and searched for correlation in the data. The findings were used in conjunction with other studies of the architecture and dependability of the system to define an initial configuration for delivery to the customer. With our current measurement programme, we would also wish to publish, via the process owner, recommendations for the handling of such error prone components. These could include design recommendations to delimit module size, coding recommendations to delimit acceptable complexity measures, or quality recommendations to systematically follow a certain level of regression testing when the number of modifications to a source file reaches a pre-determined limit.

210 Software Quality Management

2.4 Quantifying our maintenance requirements

The technical division of RED within which our pilot project is being developed is organised on a matrix basis, with 3 cross-project teams responsible respectively for specification, development, and integration. Within RED there is a further department (SQM) responsible for independent validation and maintenance of products and systems delivered by each of 4 technical divisions. Of the indicators identified in §2, those numbered 6-8 are particularly concerned with tracking product quality via anomaly reports issued during system validation and operation. The SQM department is currently responsible for tracking the backlog of outstanding anomalies via a summary sheet such as that shown in figure 5.

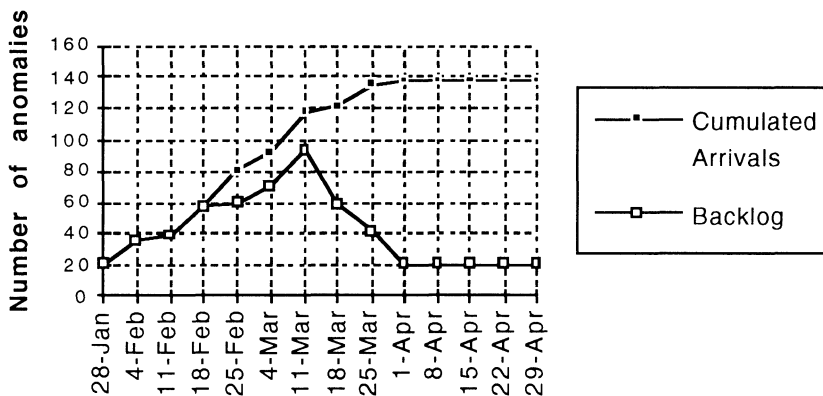


Figure 5 : Tracking reporting and fixing of anomalies.

This graph is used by the maintenance manager to help manage resource assignment between the numerous systems, products and releases which are to be maintained. Releases of future versions can also be planned with reference to the known levels of residual defects and projections for their future detection in the field. The data shown in figure 5 refers to a PLC programming console in an obsolete Cegelec product line.

2.5 Adapting the MODAL Documentation

Because of the widespread impact of the chosen indicators upon the technical and managerial processes embodied in the MODAL life cycle, a significant amount of work is involved in updating the MODAL documentation to reflect the new working practices implied by the measurement programme. Since we (correctly) expected our approach to be modified by the feedback received from the pilot project, we decided to amend the documentation in two stages. We firstly prepared a single document which gathered together all of the information which we had developed on our measurement programme. This included an explanation of the chosen indicators and their exploitation by the various roles defined within MODAL. We described mechanisms for consolidation of data at the project, divisional and corporate levels and developed a data model for its representation. Finally, we identified elements which were required to be added to existing MODAL documents (reports, minutes etc.) in order to gather all of this data. The second stage in documenting the programme will be undertaken once the results from the pilot project are complete and the fine tuning of our approach finished. Because of the large distribution of the documents and the

need to maintain coherent versions in English and French, MODAL documents are maintained via a strict change control procedure enforced by an Editorial Advisory Board. Change requests will be submitted to the board for modifications to many documents, including:

- procedures for quality assurance, project management and software maintenance,
- recommendations for quality control, project estimating, project monitoring and code peer review techniques,
- templates for the project management file, project summary, software quality plan and software quality log.

2.6 Providing Tool Support

In order to facilitate the kicking off of the measurement programme, we wanted to provide projects with a lightweight tool set which did not provide any unnecessary constraints on the management or development environment. We have therefore limited tool support to the following:

- a program to count the number of source instructions (see §3). For the pilot project, a first implementation will count 'C' code instructions and execute upon the Unix[®] environment used for configuration management on the project.
- the set of summary sheets supporting analysis of the indicators are provided using Excel[®]. An in-house tool, GEMO, is used to implement a rigorous change management cycle upon projects within RED. Specifications have been written for output by the GEMO tool, in Excel format, of defect tracking data to be consolidated by the quality manager prior to publishing the set of summary sheets.
- the preferred tool for project management included in MODAL is OpenPlan[®]. We are currently exploring ways of directly gathering into our Excel environment effort figures logged under OpenPlan.

3. OUR PILOT PROJECT

In selecting a pilot project upon which to try out our approach, we were very mindful of the need to clearly define our objectives before choosing between the various candidates. These objectives could be summarised as follows:

- we needed to validate the practicality and usefulness of the indicators which we had chosen. Although we had sought industry-proven measures, we were not sure whether they would prove to be workable or relevant in the context of our organisation, and upon the types of project which we typically undertake.
- we were aware of the need to detail the counting rules for a number of the principal data elements. In our initial document, we had outlined that software size would be measured by source volume (as opposed to, for example, size of object code) and that our preferred measure was the number of shipped source instructions (e.g. not counting harnesses, test cases etc). This was essentially a pragmatic decision to avoid extensive and inconclusive debate on the "best" measure by deciding to follow a measure extensively used in the literature and which would facilitate comparisons of our own defect densities with other published findings (assuming that defect counting rules were similarly compatible). The notion of source volume was well understood within the company, as it was also used as the basis of our effort estimation model. The recommendation which we developed prior to the pilot project, however, did not include detailed rules for counting instructions. Again, we

212 Software Quality Management

chose a pragmatic approach, having access to rules published in the literature for a number of programming languages. A tool for counting 'C' code instructions already existed which we believed could easily be adapted to support these rules. Even more critically, we had not thought deeply about developing taxonomies of defects originating in the upstream activities of specification and design. These we intended to provide "on the fly" following input received from the project team and our experiments in designing and gaining acceptance for various data collection forms. Some of the problems which this approach led us into are described in section 4.2.

- we further needed to validate the low level of tool support which we provided, to ensure that the tools were sufficiently reliable and adaptable to be widely used upon future projects.
- we wished to seize the opportunity to initiate a training programme intended to publicise our work as widely as possible within the organisation and kick off focused discussion on the topic. Presentations to the RED Quality Board and the meeting of Methodology Correspondents revealed that many people were interested in our ideas but were often sceptical of our ability to put such a programme in place and were waiting for reports of real experiences before demanding help in setting up similar initiatives.
- obviously, we would also use the pilot project to provide us with initial data for our RED data base.

We chose a project undertaken within one of the departments of RED which we felt to have a particularly mature development process and where management support for quality initiatives in general and MODAL in particular was high. This department is responsible for integration of small systems from existing Cegelec products or new components developed by other RED departments (programmable logic controllers, industrial networks, process control and supervision stations). The project is a typical one for the department in question, although it would be considered very small for many of the other application divisions, employing up to 6 people over a period of 20 months and delivering a system comprising some 25,000 lines of code of which 15% is embedded on board a programmable logic controller. We believed this project size to be optimal, considering the desire to follow a real project throughout its entire development cycle, while minimising the length of the loop feeding experience back to the measurement programme. The chosen project was to develop a distributed client-server communications architecture for process control applications. This provides an interface to "client" users of one or more process control and supervision stations which allows them to subscribe to process variables. These variables are managed transparently by applications executing upon a network of programmable logic controllers, each of which contains a "server" element, responsible for detecting and transmitting changes in the state of subscribed variables. A third sub-system (the "generator") allows the clients and servers to be configured from existing Cegelec consoles.

The development closely follows MODAL, with the life cycle shown in figure 6 (only activities treating those work products providing defect data to the measurement programme are shown). In addition to the three "generic" sub-systems cited above, specific "adaptive" components are developed to interface with the control posts and PLCs chosen for the final installed configuration. These small programs follow a similar, though somewhat simplified, life cycle.

A rigorous system of quality control, based upon use of author / reader cycles (Marca⁸), was followed for all specification documents (SSD - System Specification Document and ISD - Interface Specification Documents), design documents (PDD - Preliminary Design Documents) and end user documents (IM - Installation Manual and UM - User Manual) as well as for other documents such as quality and management plans, and test plans. Formal code reviews were followed for all software components and test reports completed at the end of each test phase. From integration testing onwards, formal reports were completed for each anomaly detected, and included in the Integration Test (IT), Validation Test (VT) and Acceptance Test (AT) Logs.

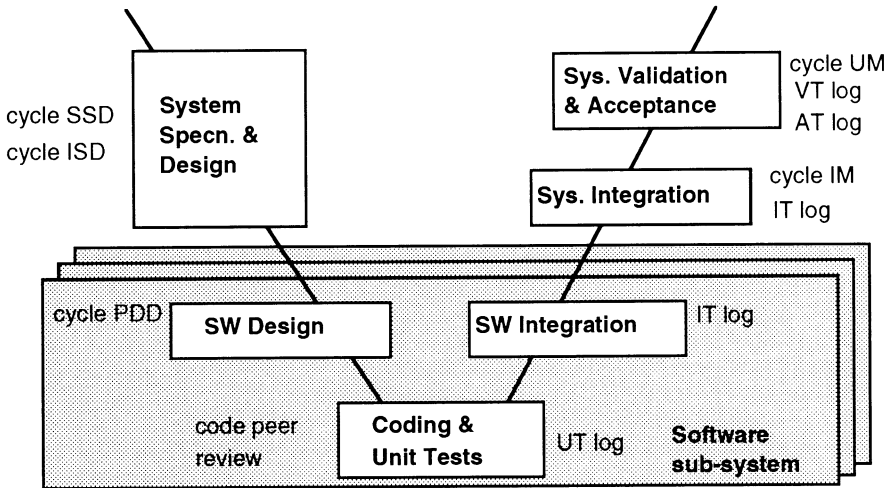


Figure 6 : Pilot project life cycle showing defect measurement data sources.

4. PROBLEMS ENCOUNTERED ALONG THE WAY

4.1 Defining our environment

In developing our set of indicators and summary sheets, we had a rather simplistic vision of "the project" which encompassed development up to delivery to a customer and subsequent maintenance support during operation. This model is restricted in that it fails to take account of maintenance or version development after the first delivery (except in terms of anomaly reporting and defect fixing via the indicators numbered 6-8) and is difficult to apply to projects intended to port existing systems or which intend to otherwise reuse large elements of existing systems. In the case of our pilot project, we had difficulty in defining the limits of the current project, due in part to the failings in our model of the project scope. While an initial system was to be delivered to an application division specialising in engineering of industrial processes, for installation at the site of the ultimate customer, the French Electricity Generating Board, an architecture was requested which would clearly separate those components able to provide generic services, reusable in similar future systems, from those which were required to interface with specific PLCs, control and supervision stations or programming consoles.

214 Software Quality Management

For the initial system, the generator and client software were to execute under VMS in conjunction with VAX station implementations of a Cegelec supervision station and the configuration software for a PC-based programming console. The server software was to be embedded in PLCs connected by Ethernet to the VAX stations and PCs. All of the software was to be written in C and developed under Unix. The generic software uses the services of libraries which encapsulate operating system and communications functions in order to remain independent of the computer hardware upon which they execute and the physical network across which they are connected. We finally decided to include within the scope of the project:

- development under Unix of the generic software
- implementation of the OS and network library services upon Unix (development environment) and VMS (target environment)
- development under Unix of the interfacing software necessary for the initial delivered configuration
- porting of the generic and interfacing software onto VMS.

4.2 Defining defects

One of the cornerstones of our measurement programme was clearly the ability to measure defects in our software. Since we wished to focus our attention upon the development process, we needed to evaluate defect levels throughout the development life cycle, and decided upon an initial 4 way classification based upon the origin of the defect (specification, design, code, user documentation). For certain of our indicators (particularly those concerned with customer use of the delivered system), we needed rather to focus on anomalies. Our MODAL documentation provided us with definitions which allowed us to clearly distinguish these concepts:

- **anomaly** : disparity between expected and observed behaviour.
- **defect** : discrepancy between entry and exit work products of a software development phase, with the result that the software lacks required characteristics or is ill-suited to its mission.

While the definition of anomaly requires some interpretation to support its use in the early stages of the development process before an executable model of the software exists, the definition of defect can equally apply to more abstract software models represented by specification or design documents. However, in attempting to collect defect data from the pilot project specification documents, we were led to refine our collection procedures to reflect a deeper understanding of the nature of defects. As shown in figure 6, defects were to be identified during the specification and design phases from a classification of the remarks formalised during document author / reader cycles. The SSD document was subjected to 4 author / reader cycles over the course of 5 months, with the customer being involved in the final cycle and subsequent review. The first two cycles were internal to the specification team and operated upon incomplete versions of the document. We decided that as our defect counts should be made upon the basis of a complete and relatively stable reference, we would use only the final cycle before review (plus the review itself) to assess defect levels.

As we needed to be careful, in publishing figures, to distinguish the number of remarks made from the number of defects implied, our first task was to classify a given set of remarks. The procedure which we developed firstly



excluded from this list of candidate defects, remarks falling into the following categories:

- **duplications** : where the same remark is made by more than one reader.
- **presentation** : spelling or grammatical errors, suggestions for rewording or reordering.
- **not applicable** : remarks which do not lead to a modification of the document. Often, these are simply demands for clarification or else suggestions for changes which are rejected (and justified) by the author.

In order to identify further duplications, we then classified the remaining remarks using a matrix which distinguished vertically between criticisms of completeness, consistency, accuracy and appropriateness, which seemed to be general types which could be identified in defects regardless of their origin. The horizontal axis of this matrix identified categories of specification defect, which we considered to represent the principal elements of a requirements specification. These were the description respectively of functional requirements, execution constraints (e.g. performance requirements), required data, operational modes, interface requirements and quality constraints (e.g. dependability requirements). Use of the matrix helped us to judge whether remarks classed in the same element were really unique, or in fact referred to the same fundamental problem. We were able to identify similar categories for defects originating in design, code and user documentation, to provide a broad overall classification scheme.

Amongst the unique, valid remarks which were retained in the matrix, we next attempted to identify those which represented Change Requests (generally demands for extension or simplification of the requirements originally discussed with the customer). After removing these cases, we were left with a homogeneous group of remarks which we considered represented defects in the specification. These were then classified as being of either Minor or Major gravity, in keeping with our wish to distinguish these cases.

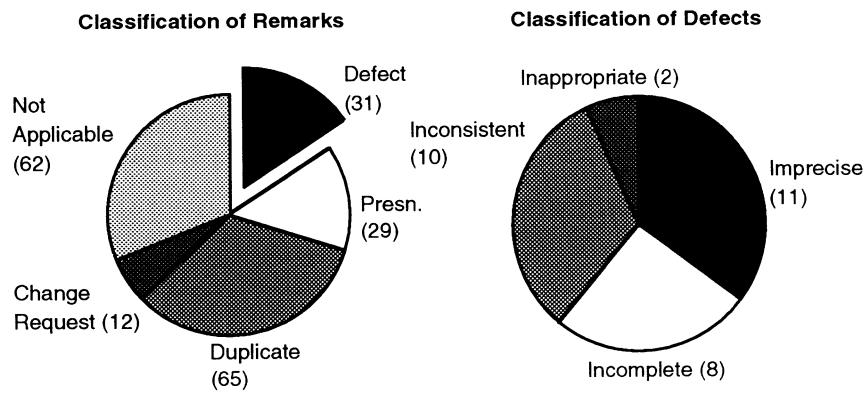


Figure 7 : Analysis of remarks on System Specification Document.

MODAL defines a major defect as one which affects the general mission of the system and which is translated by a loss of functionality, degradation in performance, loss of security or of availability. This definition too is clearly

216 Software Quality Management

oriented towards an executable model of the software, however we were able to interpret it satisfactorily for our specification documents. An example of a major defect detected in the SSD was the failure to specify performance requirements for the broadcast across the network of the time used to synchronise servers. Minor defects included inconsistencies between successive groupings of functions, and superfluous states in descriptions of the dynamic system behaviour. Figure 7 summarises the data collected from the final author / reader cycle and review of the SSD, covering 199 recorded remarks and 31 identified defects.

4.3 Collecting data

It must be stressed that throughout this classification process, the opinion of the document author was actively sought and taken into account. As we had not completed our defect classification scheme at the time at which the final author / reader cycle was performed on the SSD, no attempt was made to capture the classification information during the cycle, but rather the quality manager had to examine all of the completed forms and classify them individually, interviewing the reader and / or author where necessary for clarification. In order to reduce the overhead introduced by this procedure, it was replaced for author / reader cycles of subsequent documents by adapting the data collection forms to allow for an initial characterisation by the reader of the origin, type and gravity of each remark which he believed to reveal a defect (this was not necessary for remarks falling into the "presentation" category). These were then accepted or amended by the author and returned to the reader with the rest of the replies to each remark. The project quality manager was to act as arbitrator if no agreement could be reached between the author and reader. The standard MODAL form used for author / reader cycles was adapted for use on our specification, design and user documents, by adding the required fields to effect the necessary data collection. On the cover page the reader was required to include the effort spent in reading and criticising the document and the author was asked to specify the time spent in replying to the remarks. Elsewhere, columns were added to allow each presumed defect to be detailed by origin, type and gravity. We further adapted these forms for use upon code peer reviews, including fields to capture the size of the component reviewed.

Once code reviews have been held and executable files are available, our sources for defect data become the test logs and associated anomaly reports (see figure 6). We preferred to use summary information of the anomaly reports produced by the GEMO tool rather than examine test logs for those development phases during which the tool is used (integration and validation). We are however required to examine test files for the unit test phase, as anomalies detected during this phase do not lead to issuing of an anomaly report. MODAL however does require unit tests to be designed and documented in a Unit Test File, where the results of their execution (number of test cases passed / failed) are subsequently logged. We have amended the template of this latter document to provide summary information of the origin and gravity of each defect found, although no visibility is provided of individual defects. This approach was a reasonable compromise between imposing full anomaly tracking during unit tests (considered to be an unacceptable overhead for the developers who also act as unit testers) and not collecting any data pertaining to the unit test phase (which would nullify the attempts to study, via indicators numbers 2-4, the relative efficiency of testing and quality control activities).

We had a similar choice of sources for the data relating to effort, which we wished to investigate particularly with respect to two of our indicators. The effort estimate accuracy (indicator 1) requires the total consumed project effort to be allocated per phase. A work breakdown structure (WBS) is used to develop the project plan and resource consumption is tracked against activities of this WBS via weekly progress meetings. By making the project phases visible in the upper levels of the WBS we can roll up effort data and allocate it to individual phases. The formal end of phase review which verifies the presence of all work products output by the phase can then be used to present this data. We adapted the forms used for the minutes of this review to allow the capture of this data.

The defect detection / correction effort (indicator 2) requires figures for the average effort spent, for a given phase, in detecting and correcting defects of a given origin and gravity. For defects detected during integration and validation testing, the anomaly reports, analysis sheets and modification reports managed by the GEMO tool already provided the necessary information. Unit test reports provided overall effort and defect numbers which allow the calculation of an average figure. Adaptation of the collection forms for defects detected by quality control activities (author / reader cycles and code peer reviews) has previously been described. Similar adjustments were required for capture of the code review effort required in assessing code review efficiency (indicator 4). While we felt that the effort recorded on these forms was generally accurate we had problems in globally validating our data, in knowing whether or not this effort was reflected in the figures used for the effort estimation indicator. Using the OpenPlan tool, the granularity of resource attached to an activity was a half man-day, so that while separate activities were systematically identified in the WBS for reader cycles or code reviews, this was essentially used to aid in scheduling these activities, and effort consumed was not systematically charged against these activities. While confidence could be placed in the data used for indicator 2, it seemed that the effort used for indicator 1 was likely to lack a significant amount of work involved in quality control. As the long-term exploitation of this indicator is essentially to help provide better estimates for future projects, this seemed to be a serious problem, likely to jeopardise the usability of the indicator. We finally decided, for the purposes of calculating indicator 1, to sum the effort charged against development activities of the WBS with effort for the quality controls calculated from the data collection forms.

5. CONCLUSIONS

We believe that the experiences which we report in this paper provide a promising start to our ambitious programme to introduce software measurement into the MODAL development methodology. With the pilot project towards the end of the design phase, we have limited results to present in terms of using the defined indicators to manage projects and control quality levels, however we have refined our initial recommendations so as to provide concrete, readily-applicable procedures to future projects for data collection and exploitation.

Equally importantly, we have made significant progress in convincing senior staff and management of the benefits which can be obtained from the implementation of this programme. Once the pilot project has terminated in April 1994, our recommendations will be republished in the MODAL documentation and incorporated into the training material delivered by the Method Support

218 Software Quality Management

Department. A centralised measurement unit will be set up within this department to aid application divisions in applying the measures to their projects and gathering data to consolidate divisional data bases. Once the process owner role becomes fully operational within each division, the corporate know-how represented by the data base will be translated into a management handbook, containing guides on adapting MODAL to the development maturity level practised within the division. As a second step, we will seek to provide detailed guidelines for distinct project types, by distinguishing important characteristics of a project in the data gathered. This could initially be done using the coarse distinctions made by the COCOMO model (organic, semi-detached and embedded), as these notions are already familiar to our project managers. Finally, we could extend our concept of software size to encompass upstream documents such as specifications and designs, in order to more closely analyse the development and quality processes associated with these documents. Software size is currently used merely to normalise defect levels and provide defect densities which can be compared between projects.

We are fully committed to the movement behind development process improvement, and see it as playing a vital role in improving the quality of our products and systems, by allowing us to detect as early as possible the defects which are generated, and to make deliveries to our clients in which an estimable minimum of defects remain. By providing an increased visibility of our development process, it is hoped that the measures outlined in this paper can help us to formalise our corporate experience and aid us in implementing a truly repeatable high-quality process.

REFERENCES

1. Humphrey, W.S. *Managing the Software Process* Addison-Wesley, Reading, Massachusetts, 1989.
2. McCabe, T. 'A Software Complexity Measure' *IEEE Trans. Software Engineering*, vol. 2, pp. 308-320, December 1976.
3. Pham Van, N. 'MODAL ou l'analyse de la fiabilité intégrée dans une méthode d'organisation pour le développement d'applications logicielles' *La lettre de la sûreté de fonctionnement*, numéro 29, pp. 3-6, janvier 1993.
4. Forse, T. *Qualimétrie des systèmes complexes, Mesure de la qualité du logiciel* Les éditions d'organisation, Paris, 1989.
5. McCall, J., Richards, P. and Walters, G. *Factors in Software Quality*, 3 vols., NTIS AD-A049-014, 015, 055, November 1977.
6. Boehm, B.W. *Software Engineering Economics* Prentice Hall, Englewood Cliffs, NJ, 1981.
7. Jones, C.L. 'A Process-Integrated Approach to Defect Prevention' *IBM Systems Journal*, 24(2), pp. 150-167, 1985.
8. Marca, D.A. and McGowan, C.L. *SADT* McGraw-Hill, New York, 1988.

The following names are registered trade marks : MODAL (Cegelec), Excel (Microsoft), Logiscope (Verilog), OpenPlan (WST Corporation), Unix (AT&T).