

Quality management and end-user fourth generation systems

J.B. Williamson

Bolton Business School, Bolton Institute, UK

ABSTRACT

This paper will first discuss the some of the general principles involved in the management of quality within the context of software development. The nature of fourth generation systems is then analysed, observing their past and current functionality. The location of quality in such systems is then investigated, and a set of metrics proposed and justified. The result of applying the metrics to four specific fourth generation systems is then presented. This leads to the development of a life-cycle model which is proposed as a means of ensuring software quality when the development is undertaken by end-users.

INTRODUCTION

In an information-based society, the consumer awareness of quality becomes more significant. As Croucher [3] remarks

" one need only look at the car or photographic industries to see how much perceived quality affects market performance "

The TickIt guide [5] to software quality highlights a basic problem thus :- "there are currently no universally accepted measures of software quality". It has been noted [19] that quality has always been easier to notice in its absence rather than its presence. If an organisation wishes to register under the ISO 9001 regulations then they must fulfill the following regulations including

"A management representative needs to be nominated who will be responsible for all matters affecting the quality system. His responsibilities and authority have

298 Software Quality Management

to be defined and his position in the company structure shown on an organisation chart. Although this person acts as a focal point for quality matters, the whole workforce contributes to the overall quality of the products or services. All quality procedures have to be documented and the responsibilities of individuals clearly stated."

One of the fundamental aspects of quality is that its presence (or absence) should be detected by some-one who is not a developer, and who is sufficiently empowered. As Ince [11] comments

"for example, it is highly unlikely that a recent graduate employed in a quality assurance role would be allowed to control the computing activities of a senior accountant"

From this we can see that quality control is essential, and also that quality cannot be added; it has to be built in. In order to build in quality, we need to be aware of the stages in the process of software development. A software development project should be organised according to one of the several life-cycle models.

LIFE-CYCLE METHODS

There are many different methods of organising software development and the component parts of each method are not the same. SSADM [21] consists of

Feasibility Study, Requirements Analysis, Requirements Specification, Logical Systems Specification and Physical Design.

ITEM [4] details the following stages

Information Strategy Planning, Business Area Analysis, System Design, Construction, Transition, Production.

The manner in which these stages are organised into a life-cycle is very important to the quality assurance. The Waterfall [18] category of life-cycle reflects methods which ensure that one stage does not start until the previous stage has been demonstrated to be correct and accepted as such by the user. An alternate view is that of the Spiral Model [1] which allows users to see working components, or prototypes before they accept the design. This means that some prototyping/programming effort must take place in order to ensure that the user is satisfied with the underlying concepts of the over-all system design. Gilb [7] extends the concept of the spiral concept by suggesting that usable increments are handed over to the user in increments of 1% to 5% of the project total budget. The prototype approach takes into account that users often have limited experience with modern systems. Flaatten et al [6] describe the problem thus

"When faced with a design on paper - screen layouts, English language description of procedures, flowcharts and data flow diagrams - the users may sign off on the design without realizing its impact. They may then object to the implementation later on, saying that what they had approved was different or that they had understood the design differently."

This illustrates the fact that the manner in which quality assurance is carried out defines which life-cycle model best represents the practises and procedures being used. The necessity is for quality to be assessed at each stage and this implies certain expectations. These were put into four categories by Flaatten et al [6]

- a) the project team meets its commitments (budget, time etc)
- b) the delivered system meets the specified functional requirements completely and accurately
- c) the system has quality attributes such as usability, reliability, availability, performance, security, responsiveness to problems and maintainability
- d) the system delivers the benefits that justify the development

Many of these expectations are not located in the software, but in the means of its production. The use of quality assurance techniques during development ensures quality in all aspects of the software product.

The process of software development depends on the use of methods and tools. Brooks [2] high-lighted several problems and described two sets of innovations which affect the process of software development. Firstly the use of high-level languages supporting object-oriented data constructions which allow the "program as a model" to deal with representation issues. The second set of innovations include

- buying sufficiently general ready-made software instead of having it tailor-made
- refining the requirements iteratively with the client, using increasingly better prototypes
- enhancing the design in an iterative top-down fashion adding lower level details at each step

The concept of buying in "sufficiently general ready-made software" is central to the development of commercial software. This domain includes applications where the specification of functionality is government controlled (eg payroll) or defined by

300 Software Quality Management

professional bodies (eg accountancy). Packaged application software uses installation parameters to control their *modus operandi* and the user can also specify screen designs and report formats. The concept of generic code using parameters underpins the fourth generation of software.

FOURTH GENERATION SYSTEMS

Any input to a computer which controls how some program will perform at some later date is part of software development. The syntax of the input can be represented in some notation. Whitehead [22] pointed out the significance of the notation,

"By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race."

One problem of many of the fourth generation products is the lack of an underlying notation and the syntactic/semantic ambiguities which arise. There is no clear distinction to be made between the terms fourth generation systems, languages, environments, techniques, tools etc. Martin [15] listed a set of principles involved in the design of fourth generation languages

- Minimum work - little effort required to write system
- Minimum skill - avoid expensive training
- Avoid alien syntax/mnemonics - easy to use language
- Minimum time - fast application development
- Minimum errors - automatic early error detection or avoidance
- Minimum maintenance - change should be easy
- Maximum results - applications developed should be powerful

Grindley [8] described fourth generation languages as non-procedural in nature. Pressman [17] says

"the term fourth generation techniques (4GT) encompasses a broad array of software tools that have one thing in common; each enables the software developer to specify some characteristic of software at a higher level... There is little debate that the higher the level at which software can be specified to a machine, the faster a program can be built."

There is an overlap found between fourth generation systems and earlier application program generators (APG's). The features of APG's were categorised by Lobell [13] as follows

they assist in application development as an alternative
or in addition to procedural high-level languages
their approach is largely non-procedural

they are concerned with improving the efficiency and
productivity of application development and maintenance

they are designed to be used by a spectrum of users

Although the application generator outputs third generation language in contrast to fourth generation using parameter driven run-time systems, they allow the system to be developed in a similar manner. Recently Meehan [16] suggested the following list of characteristic components of a fourth generation language

database management system, data dictionary, query language

report generator, screen definition, graphics facility

decision support (spreadsheet et al), application development

other general purpose tools eg communications

One justification for APG's and fourth generation languages is the increase in speed of program development. Martin [15] reported an experience at Playtex using ADF (an APG)-

" for the types of application shown, there is an eighty
to one improvement in productivity of application
creation. However, there were other applications at
Playtex for which ADF was not used because it did not
fit the application well "

Martin suggests that a ten to one improvement over COBOL is a more typical figure. APG's are capable of being used in order to support the rapid prototype/evolutionary design (RP/ED) life cycle. Boehm [1] describes this as follows

Use the application generator to develop a rapid
prototype of key portion of the users' desired
capability

Have the user try the prototype and determine where
it needs improvement

Use the application generator to iterate and evolve
the prototype until the user is satisfied with the
results

302 Software Quality Management

If the performance is adequate:- keep it, maintain it. If the performance is not adequate, use the prototype as a specification and re-develop.

USER INVOLVEMENT

The involvement of the user is seen to be a means of ensuring that software meets their needs. Lanning [12] points out

"With the help of experienced guides, users can provide strategic design information and participate with the professional designers on a design project. If users help direct the collection of background information, it is less likely that the professional designers will overlook important information, possibly avoiding costly re-design efforts."

Extending the role of users involvement is not new. The user may be asked what is needed, participate in design, comment on prototypes, build prototypes or undertake development themselves. Martin [15] reports on a project undertaken by end-user developers at the Santa Fe Railroad Corporation using Mapper.

"This was the beginning of what was to become one of the world's spectacular success stories. Almost every clerical function in Corville was computerised in 17 months by an end-user team of never more than four full-time people."

The end-user developer is becoming a more frequently found role in the development of corporate information systems. The problem of deciding who should be involved in software development and what their role should be was looked at by Tagg [20] as shown in Table 1

Table 1

Applications	Level of predefinition	Level of sharing of resources	Software engineering tools	Who/How
Specialist	high	variable	3GL, IPSE	Specialists
General DP	high	high	RDBMS, 4GL IPSE, work-benches	joint specialist/ user
Decision support/ expert	low, except for house-keeping	mixed shared/ private	4 GL and beyond, stats packages, ES shells	skeleton by specialists or package, rest by user with help
General office and information retrieval	High for standard skeletons, but many user parameters	largely private but sharing of skeletons and some data	standard packages + prompted options, 4GL	Package + user DIY
Engineering	'toolkit' approach- pre-defined tools used in unpredictable way	high sharing of tools, data sharing varies	4GL, ES shells OOPS, object-oriented DB	tools by specialists or package, usage by user DIY

The evolution of end-user computing has been described by Huff et al [10] as having the following stages

- 1 Isolation:- a few pioneering users struggle in a world without formal support, to experiment and learn about the technology. Applications serve more to promote understanding than to perform substantial work-related tasks.
- 2 Stand-Alone:- end-user applications are becoming part of user's job activities, and end-user dependence is observable. Data is passed from one application to

304 Software Quality Management

another via manual re-keying.

3 Manual Integration:- end-users exchange substantial amounts of data and/or programs with other end-users. Data transfer is not integrated with the applications.

4 Automated Integration:- applications are developed by users that employ effective automated connections among systems and databases of all types- corporate or end-user developed, mainframe or micro-computer based.

5 Applications are written to access procedures or data without concern for their physical location by simply describing relationships between data and transactions.

QUALITY AND THE RP/ED LIFE CYCLE

It has now been established that the RP/ED life cycle can describe end-user software development using fourth generation systems (4 GS) and that the user can develop their own skill level over a period of time. Any technique of quality assurance requires to meet Boehm's two aims:-

To develop the right product

To develop the product right

Since the end-user is developing a system for their own needs, this should ensure that they develop the right system. However, as users progress to exchange programs and data, there will be a need to ensure that the product is right for other people.

'To develop the product right' requires that the inputs to the 4 GS are correct, and that the 4 GS can support the complexity of the application. One of the main inputs to the 4 GS during development is the specification of the tables. If the application being developed involves the use of more than one table then it is important that the developer recognises the concept of data normalisation. If end-user developers do not attend a training course (often as expensive as the software), then they are dependent upon the documentation and the examples supplied with the 4 GS. Therefore, one set of metrics attempts to measure to what extent the user can develop an understanding of the need for data normalisation as well as the techniques.

One technique to integrate project development with training has been documented by Lye and Clare [14]. They describe the role of an expert as facilitator:

Embracing the role of a facilitator, as well as that of an expert, the approach promoted an ongoing environment for the

analyst/developer and users to interact and participate in the determination and implementation of an agreed solution to the identified problem situation. This not only ensured the acceptance and ownership of the system being developed, but also provided an environment for the progressive development of the users.

Many 4 GS environments allow the user unconstrained access to the contents of all tables in the database. The ability to design a screen may be associated with the facility to apply domain integrity checks. One fundamental purpose of any application is to protect the system from garbage being keyed in. It is important that a screen which handles data from more than one table can perform referential integrity checks as well. Other inputs to the 4 GS include specification of queries and reports. If these are to access data in more than one table, then the nature of the link must be specified. Therefore another set of metrics is attempts to measure the facilities of the 4 GS which allow the user to construct a solution which apply integrity tests to data as it is input.

METRICS FOR QUALITY ASSURANCE

Some 4 GS's use screen layouts as a base for table definitions, though most depend on the table being defined before screen layout can start. It is normal to indicate which field, or combination of fields, will be used as a unique key reference. Some systems allow other fields to be used as an index for the purpose of faster access. The facility to specify that a field in one table relates to a field in another table is a means of indicating the result of data normalisation to the DBMS. If the facility to enter a relationship exists, it is beneficial that the other associated tools (eg screen painter, report generator, query processor) can use these pre-defined relations.

The set of metrics which follow indicate to what extent the 4 GS will allow the development of a system which applies domain and referential integrity testing to the data input and also whether the documentation and examples encourage the user to include such features. The set of metrics has been selected as a result of the author's teaching of various 4 GS's and observing problems encountered where end-users have not been given the relevant education.

PRODUCT-BASED METRICS

- 1) Can a field, or group of fields, be defined as unique
- 2) Are relationships between tables defined outside other tools
- 3) Can two-table forms, with a one-to-many relationship, control referential integrity as data is input/maintained

306 Software Quality Management

- 4) Can data input to more than two related tables be controlled via one form
- 5) Is foreign key validation supported (look-up or pick-list)
- 6) Does query interface relate tables using pre-defined relations
- 7) Does the report generator relate tables and allow foreign key look-ups

DOCUMENTATION-BASED METRICS

- 8) Is domain integrity described
- 9) Is referential integrity described
- 10) Is data normalisation described

EXAMPLE-BASED METRICS

- 11) Do examples use tables with normalised data
- 12) Do examples represent a problem capable of solution without programming

THE PRODUCTS

- A Released four years ago, 3rd major upgrade, DOS-based.
- B Released two years ago, 4th upgrade, DOS-based, UK product
- C Released two years ago, 4th upgrade of DOS based product
- D New version of Product C, re-written for Windows
- E Released one year ago, new product for Windows

Managing Quality Systems 307

Results	A	B	C	D	E
1 Key Unique	No	Yes	Yes	Yes	Yes
2 Relations defined	No	Yes	No	Yes	Yes
3 Two-table forms	No	Yes	Yes	Yes	Yes
4 More than two table forms	No	No	No	Yes	No
5 Foreign Key validation	No	No	No	Yes	Yes
6 Query interface use relations	No	Yes	No	No	Yes
7 Report generator use relations	No	Yes	No	No	Yes
8 Domain integrity	Yes	Yes	Yes	Yes	Yes
9 Referential integrity	No	Yes	Yes	Yes	Yes
10 Normalisation	No	No	Yes	No	Yes
11 Examples normalised	No	Yes	Yes	Yes	Yes
12 Problem solvable	No	Partial	Yes	Partial	Partial

CONCLUSIONS

From the above it can be deduced that only simple applications with a limited number of entities are capable of being solved in a manner which ensures data integrity checks are performed as data is input. The more modern products tend to have more features which are required from a quality assurance perspective. There is a need for a life-cycle within which quality assurance checking can take place. There is also a need for the user to be educated in the area of data normalisation and integrity issues. The author would suggest that the end-user training/education be linked to the life-cycle of development for the first system the user develops. By adopting this approach, the user can apply RP/ED techniques where each increment to the solution is preceded by relevant instruction. This approach relates to the need defined by Croucher [3]

308 Software Quality Management

" The project culture and the life-cycle model are necessary to impose a structure on the individual tasks"

End-user computing is evolving, and as Halloran [9] indicates

" During the past few years, organizations have experienced enormous pressure to change the hierarchical and function-oriented structures that are a carry-over from the industrial era. No matter what the phenomena is called- business re-engineering, transformation, business process design or innovation- one thing is clear: organizations are fundamentally changing the way we conduct business, and IT is a critical component of the change. End users must understand the company's strategy, critical processes, performance measures and how they can add value to achieve enterprise wide goals and objectives. Only by doing so can they use IT to improve business performance."

However, the need for some other party to be involved with quality assurance remains.

REFERENCES

1. Boehm (1981), Software Engineering Economics, Prentice-Hall, 1981
2. Brooks (1987), No Silver Bullets: Essence and Accidents of Software Engineering, Computer Vol 20 NO 4, April 1987
3. Croucher (1989), Quality - The Cinderella of I.T., Computer Bulletin, August 1989
4. Davids (1992), Practical Information Engineering, Pitman, 1992
5. Department of Trade and Industry, TickIt making a better job of software, 1990
6. Flaaten et al (1992), Foundations of Business Systems, Dryden Press, 1992
7. Gilb (1988) , Principles of Software Engineering Management, Addison-Wesley, 1988
8. Grindley (1986), Fourth Generation Languages Vol 1 - A Survey of Best Practise, IDPM Publications
9. Halloran (1993), Achieving World-Class End-User Computing, Information Systems Management, Fall 1993
10. Huff et al (1988), Communications of the ACM, 1988 Issue 31.5
11. Ince, Software Development, Fashioning the Baroque, Oxford Sciences Publications, 1988
12. Lanning (1991), Let The Users Design, Taking Software Design Seriously, Academic Press, 1991



13. Lobell (1983), Application Program Generators, NCC, 1983
14. Lye and Clare (1993), Facilitation: The Key to Exploitation of Information Systems, Proceedings of the BIT '93 Conference, November 1993
15. Martin (1985), Fourth Generation Languages, Prentice-Hall, 1985
16. Meehan (1990), Fourth Generation Languages, Stanley Thornes Ltd, 1990
17. Pressman (1987), Software Engineering:-A Practitioner's Approach, McGraw-Hill, 1987 18. Royce (1970), Managing The Development of Large Software Systems, Proceedings of the IEEE, WESCON, 1970
19. Smith and Wood (1987), Enginnering Quality Software, Elsevier Applied Science, 1987
20. Tagg (1987), ...from the end user angle, Computer Bulletin December 1987
21. Weaver (1993), Practical SSADM 4, Pitman, 1993
22. Whitehead (1911), An Introduction to Mathematics, Oxford University Press