

Experiences with quality on a recent software development project

C. Moxey

Project Programmer, CICS Development, IBM United Kingdom Laboratories Ltd. Hursley Park, Winchester, Hampshire, UK

ABSTRACT

This paper discusses some aspects of the development process used in a recent project on IBM's CICS/ESA transaction processing product, from the perspective of a member of the development team, with some reflections on why it is felt that this produced a high quality result.

Although the process that was used broadly followed the standard CICS/ESA development process - with, for example, inspections at all stages - the project had a number of special considerations:

- we had to be satisfied with the correctness of the design and implementation within a **concurrent** environment
- good **performance** was of paramount importance
- a proportion of the code used **authorized** system services, and this could not be allowed to compromise the integrity of the MVS/ESA operating system
- as a consequence of these factors, the resulting code was very complex, which meant that special attention had to be paid to its **maintainability**.

The paper describes techniques which were used to ensure the quality of the software within these constraints, such as:

- prototyping and code reuse
- individual responsibility for program components
- extensive incremental Unit Testing
- equivalence analysis of errors found
- function-oriented review of the completed code
- emphasis on producing complete documentation.

Trademarks CICS, CICS/ESA and MVS/ESA are trademarks of IBM Corporation.

376 Software Quality Management

INTRODUCTION

There is an expanding body of theory about methodologies that should result in high quality software, and about how the quality that they should produce can be quantified. But what really happens when people engage in the practice of developing software?

As a real example, this paper describes a recent software project undertaken by IBM UK Laboratories at Hursley, in which I was involved, and looks at our experiences in using techniques intended to produce results of the highest possible quality. The project was part of CICS/ESA, which is one of IBM's CICS family of products. CICS (Customer Information Control System) is a major product for On-Line Transaction Processing.

Since the project was staffed by some very senior developers, I was not by any means the most experienced person involved, but I was given responsibility for co-ordinating the development team, for ensuring good communication within the team, and for providing our interface to the other groups who were involved. In this role, I was concerned in selecting and implementing appropriate quality techniques.

I have decided not to identify the product, and there are thus places in which I may have over-simplified some technical aspects in order to refrain from going into too much detail.

The project, although of moderate size (about 12,000 lines of code written by three developers, or approximately 36 man months of design and coding effort), had a number of very complex aspects which meant that the risks of problems being introduced during either the design or the coding stages were very high.

Special Characteristics of the Project

Quality was a particular challenge for this project for a number of reasons.

Concurrent processing is a notoriously difficult area in which to avoid errors, and the code had to support concurrent accesses to data in memory simultaneous with updates to the data.

The fundamental purpose of this piece of work was to provide superior performance, and there can sometimes be a trade-off between efficiency and clarity when trying to optimise code for performance.

The code made extensive use of unique features of the MVS/ESA operating system, and there were some important security considerations. These two facts led to parts of the code running in an authorized state, which meant that it had the potential to compromise the integrity of the entire MVS system in which it was running, and this was yet another area where we could not afford to allow the quality of the code to be less than perfect. Also, since these two topics (unique MVS/ESA facilities and security) are highly specialised and somewhat difficult to understand, this further contributed to the complexity of the software.

The usual desire to produce software that would be simple to maintain, in terms both of fixing any problems and of introducing any future enhancements, was therefore highlighted on this project because of the complexities that arose from the characteristics that have been described.

FUNDAMENTAL DEVELOPMENT PROCESS USED

Since the project broadly followed the standard process which is used for all CICS/ESA development, it is appropriate to give an overview of that process. It will then be easier to understand where the specific actions that were taken for this project fit in. The overview concentrates on the stages of the process relevant to the development of the code, and does not cover such aspects as the phase reviews that are held throughout the project to ensure that it is still on schedule, nor does it go into the issuing and reviewing of the external specifications.

When a requirement has been identified, a solution that will allow it to be satisfied and the justification for doing so are documented and reviewed. Once accepted, the solution is expanded into a Product Level Design, which is inspected.

The process of inspection is one that is carried out at several stages during the development cycle. As implemented in IBM Development Laboratories, the inspection process involves inviting a group of relevant experts to study the item of work, typically a design document, and then to attend a meeting together. At the inspection meeting, the document is read through and any issues or concerns with



378 Software Quality Management

it are formally logged, to be resolved by the author of the document after the meeting. The author's solutions must be agreed by the inspectors before development can proceed to the next stage.

The stage following Product Level Design can either be the Component Level Design or, as was the case with this project, development may proceed immediately to the Module Level Design stage. At this stage the Product Level Design is refined down to the module level, and the design is validated by further inspections.

Each Module Level Design is then implemented as code, and the code is Unit Tested by the developer to an agreed test plan. During or immediately after Unit Testing, the code is reviewed by at least one other developer.

As each portion of code is completed, it is passed to the Functional Verification Group who test the new function provided by this unit of code, and also carry out extensive regression testing to verify that none of the old function has been damaged.

When all, or most, units of code have been completed, the product then passes into a formal test phase, which I shall refer to in this paper as System Test. It is at this stage that the code is tested as a complete entity, and is subjected to the sort of usage that would be expected in a customer situation.

A Defect Prevention Process is applied at all stages. Whenever a defect is found, it is analysed to discover the underlying cause and some actions are taken to prevent any similar defects in the future.

SPECIAL QUALITY-ENSURING TECHNIQUES USED

As explained earlier, the complexities of the software being developed meant that it was not felt to be sufficient to follow the standard development-process formula, and a number of actions were taken which were either unique to this project or which were relatively new introductions to CICS/ESA development.

In this section I intend to go through some of these techniques, to explain why we decided to use them and what benefits we felt that they gave us.

Prototyping and Code Reuse

For reasons not originally related to this particular project, a major piece of prototype work had been carried out which was relevant to our development. Some components of the existing prototype were used to form a prototype for a vital part of the new product, which gave us savings in terms of design and code effort. This prototyping was extremely useful as it enabled us to make some predictions about how various proposed designs would behave. We saw great benefits in terms of quality because the prototyped code was very thoroughly exercised, and methods of testing that had been developed for the original piece of work could also be employed to test the new prototype and later used to re-test it following its incorporation in the final product.

Since the product was based conceptually on extending a facility that was already available within existing releases of CICS/ESA, it was also possible to reuse some small parts of that original CICS code which, having been thoroughly exercised, we could expect to be stable.

Individual Responsibility for Program Components

Each developer on this project took responsibility for a particular part of the product. Since it was effectively structured into three layers, this allowed each of the three main developers to have responsibility for one of the layers.

The first advantage of doing this was that each layer required particular skills and knowledge, and we were able to closely match the skills of the developers to the skills required.

A second advantage was the continuity that it gave, since each developer saw their layer of code right through the stages of: doing the Module Level Design; writing the code and Unit Testing it; correcting any problems found by code review, Functional Verification, or System Test; writing the internal documentation and reviewing the relevant sections of the published documentation. This gave the developers a strong feeling of "ownership" which served to enhance the desire, no doubt already present, to make sure that our components were of an excellent quality. It also meant that the people writing the code were totally familiar with all the design issues. It is relatively unusual for the same individual to write both the Module Level Design and the code, and in fact, two of us had also written the Product Level Design, giving even greater continuity.

380 Software Quality Management

A third advantage was the rapidity with which we were able to respond to any problems that were found during testing and, since the original author of the module was providing the fix, we could have a high degree of confidence that it would be suitable and would not cause any unexpected side-effects.

Although developers had individual ownership, we did not work in isolation, but inspected each other's designs and code, and also took part in design discussions to understand the interfaces and to advise each other. I should mention that this project was interfacing with an accompanying project, and that the developer of that was also very much involved in the design discussions and internal reviews that we held, so that we tended to think of ourselves as a team of four rather than of three.

Extensive Incremental Unit Testing

Although I would support the opinion that it is not possible to write a poor piece of code and expect to test all the bugs out of it, I am nevertheless a firm believer in the value of thoroughly Unit Testing an item of code.

In order to make sure that our Unit Tests would be exhaustive, we wrote very detailed test plans at design time. These test plans were themselves inspected separately, and they made extensive use of tables and checklists to ensure that no tests would be forgotten. The format that we created for the test plans has subsequently been adopted for future use on all projects in CICS/ESA.

We used a number of different conditions for the Unit Testing, ranging from exercising the modules within a scaffolded environment through to running them as part of a full CICS system. A complex CICS environment was set up which included several CICS systems and a variety of resources, and this was used throughout the development of the code.

The development of this project was carried out in a staged fashion, with the function being introduced incrementally. Thus, for example, the first stage did not support concurrent access, and the last stage introduced some final optimisations and handling of rare error conditions. At each stage, a design to the module level was produced, the code was written and Unit Tested, and the code was then made available to the Functional Verification Group, who could begin to test it while we progressed to the Module Level Design for the next stage.

This meant that each increment to the function was Unit Tested, and any problems could be resolved before the next stage of complexity was introduced.

Of the errors that were found by the three stages of testing (Unit Test, Functional Verification and System Test), almost all were found during Unit Testing. This suggests that our Unit Test technique was indeed thorough enough to capture a large proportion of the errors. In fact, during the whole System Test, only one error was found in the code. It is our hope that this was the last error remaining in the product!

"Equivalence Analysis" of Errors Found

We carried out an "equivalence analysis" on all errors, by which I mean that whenever an error was found in the code, as a result either of testing or of studying the code, particular attention was paid to checking any other places where the same or similar errors could have occurred. This was an adjunct to the standard Defect Prevention analysis of the error.

As well as multiplying the value of discovering an error, since it could potentially lead to the removal of several errors at the same time, this also meant that we made frequent in-depth studies of the code, which can only have been a good thing.

Function-oriented Review of the Completed Code

After completing the majority of the code, a thorough review was carried out on the entire set of modules, working through by function rather than by module. This exercise extended over more than three months, and involved a number of people from outside the development team.

The normal procedure for reviewing code is for a colleague to "buddy check" the changes to a module as each small increment of code is produced. We felt that the approach adopted on this project was superior to this because:

- We looked at the final modules as a whole, rather than just concentrating on lines of code that had been changed since a previous stage.
- We looked at the "wood", by following the flow through the code that would be taken by each function, rather than being

382 Software Quality Management

mesmerised by the subroutines (or "trees") making up a particular module.

- We had several people at each review session. In addition to the development team, all of whom attended most of the sessions, reviewers were chosen who had some expertise in the area being looked at on that occasion.
- Having studied the code independently in advance of the session at which it was to be reviewed, the group of reviewers then met together to discuss their findings. This allowed them to generate ideas from one another, and meant that some sort of a consensus could be reached where there were questions on matters such as coding style.
- After any corrections suggested by the review had been made, the code was again checked, this time via the normal "buddy checking" procedure.

A number of other benefits resulted from this exercise:

- The code tended to be well commented prior to the review, and where the commentary was still felt to be insufficient or confusing, improvements to it were made as a result of the review.
- A good style of programming was encouraged, and a consistent one (within the constraint that the different layers of the software had rather different characteristics).
- Representatives from the Service Organisation attended the reviews. This allowed them to make recommendations to improve the diagnostic aids and the maintainability of the code, and also gave them a chance to become familiar with the product long before they would have to take over the maintenance of it. In fact, the reviews had some educational value for most of the attendees.

We held 20 review sessions, and a large number of items were logged as requiring attention. However, most of these were minor points relating to: suggestions for improved coding style or other serviceability improvements; to incorrect, incomplete or confusing comments; and to redundant code. Less than a tenth of the items were designated as actual coding problems that could in theory have led to failures of the product or to reduced performance had they not been found and removed, and only a fraction of those were serious problems. Although this might have been thought to be disappointing in view of the investment made in reviewing the code, it was certainly reassuring, and we took the attitude that the code must not only be

correct, but that the combination of code and commentary must be sufficiently clear for everyone present to understand it.

I believe that this whole exercise was very valuable and worthwhile. In addition to satisfying ourselves that the product was already of a high quality, we were able to make many improvements to the code, we found a few important errors, and the maintainability of the software was undoubtedly much enhanced.

Emphasis on Producing Complete Documentation

I have already touched on the fact that the code itself was very thoroughly commented, from summary overviews in the headers of each module down to detailed descriptions of the use of important flags in control blocks, and including explanations and warnings about any features of the design whose rationale might not be obvious.

We also augmented this with extensive internal documentation for use by those who would be maintaining the code. This differed somewhat from the typical format of such documentation in that the full design details and code flows were preceded by a comprehensive reference section showing overviews of the functions provided, the relationships between and contents of control blocks, and exhaustive information about the modules. In addition, a conscious effort was made to include as much of the reasoning behind the various design decisions as seemed pertinent. This documentation was reviewed by a number of the people who would make use of it before being issued.

Continuing with the subject of documentation, we took a very active interest in the writing of the user documentation for this product. Some radical methods were also employed here in order to make the documentation as valuable to the future users of the product as possible, but that is part of another story.

Other Factors

Early performance testing is always highly desirable, especially when, as in this project, performance is of paramount importance. Of course, we could not achieve the impossible and carry out a performance evaluation before any code had been written, but we were able to make some predictions from an analysis of the design, and also managed to collect measurements as the code was being developed, which helped us to verify and modify the design. The staging of the development that I mentioned earlier was partly devised so that we could be receptive to

384 Software Quality Management

any adverse findings that might have arisen from the early performance studies.

The use of a focused development team, with one member acting as the interface to other groups was felt by some to have helped contribute to product quality. It meant that one person (myself) was aware of all the issues affecting the development, that I presented a consistent view of the project, that everyone in the organisation knew whom they should contact, and that the rest of the team were free to concentrate fully on their development work.

In this section I have only discussed the efforts made by the development team to ensure high quality. Each of the other groups involved had their own particular input to make to the overall quality of the product, but that is not the subject of this paper.

SUMMARY

It was recognised from the beginning that this project would be particularly challenging, and that we must therefore employ some special manoeuvres in order to produce the quality that we desired.

I believe that the first step in this direction was actually made when the team of people who were to develop (or be closely involved with the development of) the project was chosen. Three of the four were certainly very experienced and skilled programmers, and although I would hesitate to count myself as contributing in any large way to the technical excellence of the team, I would like to think that my organisational abilities helped to focus us on working towards a high-quality result.

I have described in this paper how a sense of long-term ownership of the code, a concentration on thorough unit testing and on getting maximum value out of any errors found, the use of early prototyping and performance analysis, and care in providing full documentation, all contributed to the final quality of the code that the development team were able to deliver to the test teams.

At the end of the project, we all felt very good about what we had done which, although it is not a very quantifiable measurement, is perhaps one of the best indications of quality that you can get.